ALGORITHMS FOR SELF-ORGANIZING WIRELESS SENSOR NETWORKS

A Thesis
Presented to
The Academic Faculty

by

ElMoustapha Ould-Ahmed-Vall

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2007

ALGORITHMS FOR SELF-ORGANIZING WIRELESS SENSOR NETWORKS

Approved by:

Professor Bonnie Heck Ferri, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor George F. Riley, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Mostafa H. Ammar
College of Computing
Georgia Institute of Technology

Professor Douglas M. Blough
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Henry L. Owen III
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Raghupathy Sivakumar
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: April 4, 2007

DEDICATION

To my family for their encouragement and unconditional support throughout the years.

In the memory of my grandfather, teacher and mentor Nahi Ould-Mahmoud

ACKNOWLEDGEMENTS

In the name of Allah, the Most Beneficent and the Most Merciful.

By Allah's will, aid and support, the completion of this work has become a reality. I would like to start by thanking the Almighty for the uncountable bounties and blessings He had granted me. I am most grateful to Him for giving me the inspiration and confidence to pursue this work to its conclusion.

Special thanks are offered to my PhD advisors Prof. Bonnie Heck Ferri and Prof. George F. Riley. I am indebted to them for their unselfish commitment of time, conscientious advice, sound guidance, sincere efforts, constant support and encouragement. They allowed me to have freedom and flexibility in my research choices and helped me every step of the way. Their constant determination and efforts have been the guiding lights throughout my PhD years work and I whole-heartedly thank them for their trust and confidence in me.

I would like to thank Prof. Douglas M. Blough for his constant collaboration on the global ID assignment work presented in the third chapter of this thesis. I am also thankful for his serving as a chair of my PhD proposal committee and a reader of this thesis. I also would like to thank Prof. Henry L. Owen III for reading this thesis. My gratitude and thanks go to Prof. Blough, Prof. Owen and the other committee members, Prof. Mostafa Ammar and Prof. Raghupathy Sivakumar, for their valuable and timely feedback.

During my PhD years at Georgia Tech, I met many dedicated, knowledgeable and helpful faculty and staff members. Special thanks go to Prof. Yalamanchili, Prof. Copeland, Ms. Gail Palmer, Prof. Tovey, Prof. Ayhan, Prof. Tsui, Prof. Nair-Reichert, Prof. Ghosal and Prof. Boston for their highly interesting and passionately

taught courses. I thank Ms. Josyane Roschitz for her help and kindness during my stay at Georgia Tech Lorraine. I would like to express my gratitude to Ms. Sheila Schulte of the Office of International Education for her constant help and support. I wish to thank Ms. Jill Auerbach for giving me the opportunity to be a mentor for the Intel Opportunity Scholarship program. Among the ECE staff members I relied on, Ms. Marilouise Mycko, Ms. Beverly Scheerer and Ms. Leslie Hudson deserve special mention.

Throughout my stay at Georgia Tech, I have had the opportunity to meet and interact with many fellow students. Special thanks go to Tamir Hegazy, Talal Jaafar, Dheeraj Reddy, Nader Metwalli, Robby Simpson, Young-Jun Lee, Mohamed Abdelhafez, Xin Zhang, Jawad Ahmad, Mazin Mekki, Xenofontas Dimitropoulos, Amjad Yagi, Sunitha Beeram, Nurudeen Olayiwola, Elizabeth Whitaker, Nitya Sundareswaran, Mansoor Ahmed, Richelle Adams, Usman Chaudry, Mike Sun, Selcuk Uluagac, Chris Lee, Cyrus Harvesf, Faisal Shah, David Bauer, Amil Haque, Ilker Kalkan, Saad Kabir, Nicolas Gastaud, Zeechan Sayed, Jassem Abdallah, Michael Michaux, Zaib Talat, Mahmoud Elhaddad and Sameh Dardona. I also would like to thank many other friends who made my PhD years as enjoyable as possible: Imam Zahid, Toufeeq Ahmed, Srinivas Vadrevu, Mohamed Yahya, Moualoud Ould-Bouh, Motasem Saidan, Youssef Lebbar, Anirban Basu, Khaled Ould-Beya, Thomas Hecquet, Abderrahmane Kane, Mohamed Lemine Ould-Cheikhatou, Amadou Deme, Mohamed Goram, Farouq Ba, Muawiya Khreis and Ashraf Awad.

My various internships allowed me to interact with real-world engineers and enriched my knowledge and understanding of key technological issues and how research can be applied in an industrial setting. Special thanks go to Mr. James Abel, Dr. Seth Abraham, Dr. Kshitij Doshi, Mr. Antonio Valles, Mr. James Woodlee and Dr. Charles Yount at Intel Corporation for many helpful discussions and suggestions during my 2006 research internship at Intel.

Last but not least, I would like to express my thanks and gratitude to my parents, grandparents, siblings, uncles, aunts and cousins for their continuous love, support and patience. My parents have always provided me with the best education and opportunities they could afford. The completion of this thesis is a tribute to their sacrifices and hard work to educate me and my brothers. I would like to express my special thanks to my aunt Hind for being my first and in many ways most influential teacher. I also would like to take this occasion to thank Abdallahi, Mohamed-Vall, Mohamed-Mahmoud and Yakoub for their financial and moral support throughout my college and graduate years.

The research in this thesis is supported by the United States National Science Foundation under contract numbers ANI-9977544, ANI-0136969, ANI-0240477, ECS-0225417 and CCR 0209179 and DARPA under contract number N66002-00-1-8934.

Although the help of many people is acknowledged here, I must confess in all humility and sincerity that only I am responsible for the shortcomings of this thesis.

ElMoustapha Ould-Ahmed-Vall

Atlanta, GA

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

A sensor network consists of a set of nodes powered by batteries and collaborating to perform sensing tasks in a given environment. It may contain one or more sink nodes (base stations) to collect sensed data and communicate it to a central processing and storage system. A sensor node is typically powered by a battery and can be divided into three main functional units: a sensing unit, a communication unit and a computing unit.

The unique characteristics of sensor networks pose numerous challenges that have to be overcome to enable their efficient and reliable use. In particular, sensor networks are highly energy constrained because of their reliance on battery power and the difficulty and cost of battery replacement. These networks are generally composed of a large number of inexpensive and potentially unreliable individual nodes. These characteristics render the efficient collaboration between individual nodes essential to the accomplishment of the overall network task and justify the development of new algorithms to provide services such as information processing, messages routing, fault-tolerance, localization, naming and addressing.

This work increases the knowledge on the growing field of algorithms for wireless sensor networks by contributing a new evaluation tool and two new algorithms.

A new sensor network simulator that can be used to evaluate sensor network algorithms and sensor network architectures in general is discussed. The simulator incorporates models for the different functional units composing a sensor node and characterizes the energy consumption of each. It is designed in a modular and efficient way favoring the ease of use and extension. The simulator allows the user to choose

from different implementations of energy models, accuracy models, communication protocols, application classes and types of sensors. New models can be easily added if necessary.

The second contribution of this thesis is a distributed algorithm to solve the unique ID assignment problem in sensor networks. Our solution starts by assigning long unique IDs and organizing nodes in a tree structure. This tree structure is used to compute the size of the network. Then, unique IDs are assigned using the minimum number of bytes. Globally unique IDs are useful in providing many network functions, e.g. configuration, monitoring of individual nodes and various security mechanisms. Theoretical and simulation analysis of the ID assignment algorithm solution are presented. The results demonstrate that a high percentage of nodes are assigned globally unique IDs at the termination of the algorithm when the algorithm parameters are set properly. Furthermore, the algorithm terminates in a relatively short time that scales well with the network size.

The third contribution of this thesis is a general fault-tolerant event detection scheme that allows nodes to detect erroneous local decisions based on the local decisions reported by their neighbors. This detection scheme does not assume homogeneity of sensor nodes and can handle cases where nodes have different and dynamic accuracy levels. We prove analytically that the derived fault-tolerant estimator is optimal under the maximum a posteriori (MAP) criterion. An equivalent weighted voting scheme is derived. Further, we describe two new error models that take into account the neighbor distance and the geographical distributions of the two decision quorums. These models are particularly suitable for detection applications where the event under consideration is highly localized.

CHAPTER I

INTRODUCTION

A sensor network consists of a set of nodes powered by batteries and collaborating to perform sensing tasks in a given environment. It may contain one or more sink nodes (base stations) to collect sensed data and relay it to a central processing and storage system. A sensor node is typically powered by a battery and can be divided into three main functional units: a sensing unit, a communication unit and a computing unit. A large number of these sensor nodes can be deployed quickly in a sensing field where each node independently monitors its immediate environment (sensing range) while collaboratively participating with other nodes in the network to accomplish complex information related tasks such as gathering, processing and dissemination of information.

Sensor networks have potential for use in many military and civilian applications including habitat monitoring [18, 108, 58], environmental monitoring [12, 100], health systems [92], target tracking and localization [53, 74, 75]. However, the unique characteristics of sensor networks pose numerous challenges that have to be overcome to enable their efficient and reliable use. In particular, sensor networks are highly energy constrained because of their reliance on battery power and the difficulty and cost of battery replacement. These networks are generally composed of a large number of inexpensive and potentially unreliable individual nodes. These characteristics render the efficient collaboration between individual nodes essential to the accomplishment of the overall network task and justify the development of new algorithms to provide services such as information processing, messages routing, fault-tolerance, localization, naming and addressing.

This chapter gives first an overview of some of the popular application scenarios of sensor networks in Section 1.1. In Section 1.2, we describe some of the unique characteristics of sensor networks that justify the development of new algorithms. Section 1.4 introduces the main contributions of the thesis.

1.1 Sensor Network Applications

Sensor nodes can be equipped with different types of sensors such as thermal, seismic, visual, acoustic, infrared and radar. These sensors can be used to monitor a variety of phenomena such as temperature, lighting levels, humidity, presence or absence of a specific target or types of targets, object movement and pressure levels.

Due to their ease of deployment and the fact that they do not require the pre-existence of communication infrastructure, sensor networks can be used in many application areas. The understanding of these different application areas and their requirements is crucial to the design of efficient algorithms for sensor networks.

1.1.1 Health Systems

The use of sensor networks can help create a more proactive health system as opposed to the current systems designed for reaction to crises and illness management [65]. A reactive system can help in the reduction of the rising overall cost of health care in the US and worldwide [67]. Wearable health monitoring systems are a key to wellbeing and health monitoring, which can lead to early detection of disease signs allowing its prevention or early treatment. In addition, sensor networks can be used to monitor and report on the elderly and patients requiring continuous supervision.

Intel and the University of Washington initiated the Caregiver's Assistant and CareNet Display [26, 80] project aimed at enabling better care for the older patients by monitoring their activities. Sensor nodes monitor various household objects to collect and report information on which objects are touched and when. The information is used to complete the Activities of Daily Living (ADL) form by Caregiver's Assistant,

an artificial intelligent program. The collected information is also used by the CareNet Display to allow the family access to detailed activities of the elder patient.

Another interesting application in this domain is the CodeBlue project from Harvard University [34, 33]. This project developed a sensor platform that uses pulse oximetry and electrocardiogram sensors to allow continuous monitoring of patient vital signs and health condition. The collected information can be transmitted over a short range (up to 100 meters) and used to alert health providers of important changes in status.

In [93], the authors proposed the use of an artificial retina chip consisting of 100 microsensors. The system can be implemented in the human eye to allow patients with limited or no vision to see. Wireless communication is used to enable feedback control used for image identification.

Other potential applications of sensor networks in the health care field include tracking and monitoring of doctors and patients inside a hospital, collection of human physiological data and drug administration in hospitals [5].

1.1.2 Environment Monitoring

Sensor networks can be used to monitor environmental conditions such as temperature and humidity, which enables applications such as wildlife habitat monitoring, forest fire detection and flood detection. Habitat monitoring has been proposed in [18] as a driver application for wireless sensor networks.

The Great Duck Island project [102] aims at non-intrusive monitoring of the Pertel's (sea bird) nesting behavior. A sensor network is deployed to collect environmental variables such as relative humidity, temperature conditions and pressure level. These variables are collected in and around bird nests without disturbing the birds.

The PODS project [13, 14] at the University of Hawaii uses sensor networks equipped with environmental sensors and cameras to study why endangered vegetable

species are more likely to grow in certain areas than in others.

Sensor networks can also be deployed to monitor forest fires. The FireBug project at UC Berkeley uses sensor nodes to collect temperature, relative humidity and barometric pressure. This data is used in order to detect the initiation and evolution of wildfires. Early deployment [28] served as a proof of concept of the potential use of sensor networks for real-time fire monitoring. Similar projects include the deployment of a sensor network based fire protection system in Split-Dalmatia region of Croatia [56].

1.1.3 Home Applications

As sensor nodes get smaller and smaller, it becomes possible to mount these nodes on home objects such as vacuum cleaners, microwave ovens and VCRs [79]. Nodes sensing such objects can form a sensor network and interact with the outside world, for example via Internet, to allow applications such as remote management of home devices.

The Intelligent Home Project (IHome) [2] at the University of Massachusetts uses a set of distributed smart agents deployed throughout the house to coordinate shared resource utilization. Shared resources include water, electricity and heating. A cost function can be associated with the usage of each resources with sensor data used to decide levels of activation of different resources. For example, the air conditioning activation can be controlled by sensors detecting the presence or absence of individuals in different rooms.

Other home and office applications of sensor networks include the Aware Home project [1, 4] at Georgia Tech targeted at better care of the elderly and the Smart Kindergarten project [3, 20] at UCLA applying sensor networks to early childhood education.

1.1.4 Other Applications

Many other applications have been proposed for sensor networks. In particular, sensor networks have great potential for use in the military field [25] to monitor friendly forces and equipments; track and target enemy forces; assess battle damages and participate in reconnaissance operations.

Other applications include:

- Traffic monitoring where sensor networks are used for vehicle detection and classification [24].
- Structural health monitoring (SHM) [42] to detect and localize structural anomalies.
- Monitoring volcanic eruptions at a specific site [110].
- International border monitoring where a sensor network is used to detect intrusions along the US-Mexican borders [9].

1.2 Differences with Other Types of Networks

Despite the wide spectrum of sensor network applications as illustrated by the examples presented in the previous section, many proposed sensor network designs share aspects that differentiate them from traditional networks.

1.2.1 High Scalability

Sensor networks are often composed of a large number of nodes (hundreds, thousands or even tens of thousands of nodes). A network with such a large number of nodes has several advantages over sensing systems of one or few sensor nodes. In fact, sensor networks can cover large areas that cannot be covered by small-scale sensing systems composed, for example, of few wired sensors. In addition, a network composed of many nodes is likely to be more robust to node failures.

The large number of nodes in a sensor network creates new challenges. In particular, the traditional methods of configuration and deployment of computer networks are impractical in the case of sensor networks. In fact, it is too costly if not impossible to manually configure and deploy each node. In addition, due to the nature of the deployment terrain it might be more beneficial from a performance standpoint to have the nodes self-configure after the deployment.

The use of a large number of nodes makes the scalability an important factor in the design of sensor network algorithms.

1.2.2 Limited Energy Resources

A sensor node is typically powered by a battery, which makes energy efficiency a critical need for sensor networks. These networks are often deployed in unfriendly remote environments such as a forest. This isolation makes changing the battery or charging it expensive, if not impossible. For this reason, maximizing the lifetime of a sensor network while keeping an acceptable performance level has seen substantial interest from the research community.

The lifetime is maximized through innovative approaches such as new network protocols consuming less energy, smarter collaboration strategies between neighboring nodes and more efficient processor power management, just to name a few. Many of these approaches extend the lifetime by limiting the communication since it has been shown that communication is the most power consuming task in many sensor network settings [85, 64, 81].

1.2.3 Dynamic Topology

Sensor networks are highly dynamic. This dynamic nature comes from the dependence of sensor networks on their environment. For example, nodes can be carried away by wind, eaten by a wild animal or damaged by the sun's heat. Nodes can also run out of energy, experience changes in their communication range or suffer software errors.

Sensor networks are not the only type of networks that can experience very dynamic behavior resulting in frequent topology changes. The Internet is a network of many nodes organized in a highly dynamic topology. The main difference between sensor networks and other types of networks, such as the Internet, is the ratio of the number of users to the number of nodes. In fact, sensor networks have a very small number of users (potentially one user) for a large number of nodes while the Internet has a large number of users, close to one user per node. This makes it impossible to rely on users to handle problems resulting from changes in the case of sensor networks. This dynamic nature requires sensor networks to be highly adaptive and able to self-configure.

1.3 Algorithms for Sensor Networks

The unique characteristics of wireless sensor networks described above and the nature of their applications motivate the development of new algorithmic solutions to provide services such as information processing, messages routing, fault-tolerance, localization, naming and addressing.

Traditionally, algorithms have been classified as either centralized or distributed. Centralized algorithms execute on a single node, which is impractical for large-scale highly constrained sensor networks. This approach would require all nodes to send their respective information (e.g., sensed data, or energy level) to a specified location (e.g., a sink node) and wait until this node executes the algorithm and sends back its output. This approach is unsuitable for sensor networks because of its cost in terms of energy and delay. In addition, it suffers low fault-tolerance and low scalability and creates bottlenecks [31].

In the case of distributed algorithms, different nodes participate in the computation which can reduce the execution time. However, the computation at each node can still depend on information sent by nodes that are located far from it. In this

situation, we are still faced with a communication energy cost that is potentially prohibitive.

To address this problem, a new class of algorithms called localized algorithms [31, 32] has been proposed for sensor networks. Localized algorithms are a special category of distributed algorithms that are designed specifically to enable efficient and reliable collaboration between sensor nodes by taking into account their specific constraints. For instance, a localized algorithm might limit the collaboration and information exchange to close neighbors of a node to save energy. Yet, such algorithms allow the accomplishment of network-wide objectives and tasks such as event detection and monitoring.

Localized algorithms for wireless sensor networks were first introduced in 1999 by Estrin et al. in [31, 32]. This new class of algorithms has seen a growing interest from the research community in the last few years. Localized algorithms offer several advantages for wireless sensor networks. In particular, the highly localized system of interaction limits the communication, which can substantially reduce the energy cost and the execution time. Localized algorithms are also highly fault-tolerant since each node can continue to operate despite failures at other nodes in the network.

The development of algorithms have so far been mostly restricted to two main areas, namely network services (e.g., routing) and collaborative information processing [83]. Existing algorithms providing network services include directed diffusion [44], SPIN negotiation-based information dissemination [40, 51], SPEED real-time routing protocol [36], geographic routing protocols such as [112, 22] and the residual energy based routing proposed in [61]. Some of the collaborative information processing algorithms are proposed in [84, 116, 10, 96].

1.4 Thesis Contributions and Outline

This work proposes to increase the knowledge on the growing field of algorithms for wireless sensor networks by contributing a new evaluation tool and two new algorithms. A new sensor network simulator that can be used to evaluate sensor network algorithms and sensor network architectures in general is proposed. In addition, two new algorithms are proposed to provide global identification and fault-tolerant detection services to sensor networks. These three contributions are described in more detail in the following subsections.

1.4.1 Sensor Network Simulation

The highly scalable nature of localized algorithms allows the development of large-scale sensor networks. However, it is not always possible to deploy sensor networks of realistic sizes to test and validate new and existing localized algorithms. For this reason, simulation becomes a necessary alternative to study this type of algorithm. Several sensor network simulators are available and are in widespread use by the research community. However, most of these simulators lack models for important aspects of sensor network simulation such as energy consumption models and sensor accuracy models. In addition, most of the existing simulators do not scale well with the size of the network [71].

This work proposes a new sensor network simulation environment, the *Georgia Tech Sensor Network Simulator* (*GTSNetS*). This new simulator allows users to evaluate the effects of different algorithms and other architectural choices and strategies on the lifetime and performance of a sensor network. This tool can also be used to evaluate new approaches (e.g., routing protocols, cooperation algorithms) and compare them with existing approaches.

It incorporates models for the different functional units composing a sensor node and characterizes the energy consumption of each. It also has a model for a sensor

network base station and its interactions with the rest of the network. Models for sensing accuracy that have been lacking in existing simulators are also included. The simulator presented here is also, to the best of our knowledge, the only sensor network that supports the simulation of networked control systems having sensing, control and actuation capabilities.

This simulator is an extension of the *Georgia Tech Network Simulator* (*GTNetS*) and leverages its design choices for maximum scalability.

1.4.2 Global Identification for Wireless Sensor Networks

The communication range of individual sensor nodes is generally limited, and communication is often carried out in a multi-hop way, which creates a need to have a unique identifier for each node in the network in the header of every unicast packet. In fact, routing protocols need to uniquely identify the final destination of each packet as any node in the network can be a potential destination. Several routing protocols use attribute-based routing and, therefore, can use attributes as global identifiers. However, even these protocols require the existence of unique IDs at a local level. This is the case for directed diffusion [44] and geographical routing protocols such as the Geographic and Energy Aware Routing (GEAR) protocol proposed in [112]. Network-wide unique IDs are beneficial for administrative tasks requiring reliability, such as configuration and monitoring of individual nodes, and download of binary code or data aggregation descriptions to sensor nodes [30]. Network-wide unique IDs are also required when security is needed in sensor networks [46]. Several MAC protocols requiring the preexistence of network-wide unique IDs have also been proposed for sensor networks [113].

We discuss a new algorithm to solve the unique ID assignment problem. The proposed solution starts by assigning long unique IDs and organizing nodes in a tree structure. This tree structure is used to compute the size of the network. Then,

unique IDs of minimum size (number of bytes) are determined.

Nodes joining the network asynchronously can request unique IDs from their active neighbors. This mechanism is implemented by allowing the nodes that initially participate to request several spare unique IDs instead of a single ID. These spare IDs are assigned to newly active nodes with the guarantee that each ID is unique.

Theoretical and simulation analysis using *GTSNetS* of the proposed solution are presented. The results demonstrate that a high percentage of nodes are assigned globally unique IDs at the termination of the algorithm when the algorithm parameters are set properly. Furthermore, the algorithm terminates in a relatively short time that scales well with the network size.

1.4.3 Fault-Tolerance for Distributed Detection Using Wireless Sensor Networks

Distributed event detection using wireless sensor networks has received a growing interest in recent years. In such applications, a large number of inexpensive and unreliable sensor nodes are distributed in a geographical region to make firm and accurate local decisions about the presence or absence of specific events based on their sensor readings. However, sensor readings can be unreliable, due to either noise in the sensor readings or hardware failures in the devices, and may cause nodes to make erroneous local decisions.

A new algorithm is presented to provide fault-tolerance for distributed detection applications. This algorithm allows nodes to detect erroneous local decisions based on the local decisions reported by their neighbors. This approach does not assume homogeneity of sensor nodes and can handle cases where nodes have different accuracy levels. We prove analytically that the derived fault-tolerant estimator is optimal under the maximum a posteriori (MAP) criterion. An equivalent weighted voting scheme is also derived. Further, we describe two new error models that take into account the neighbor distance and the geographical distributions of the two decision quorums.

These models are particularly suitable for detection applications where the event under consideration is highly localized.

1.4.4 Dissertation Outline

The rest of this thesis is organized as follows. In Chapter 2, sensor networks simulation is motivated and the *Georgia Tech Sensor Network Simulator* is introduced. Several simulation experiments are conducted to illustrated the usefulness and features of the simulator. The simulator is demonstrated to scale well with network size. We also discuss a case study and contrast the simulation results obtained using *GTSNetS* and previously published results.

Chapter 3 is dedicated to the global ID assignment problem. A novel distributed algorithm is proposed to tackle this problem. The algorithm performance is evaluated through theoretical and simulation analysis. In particular, properties such as termination, correctness and energy cost are studied. The proposed solution is also extended to cover the case of asynchronous wake-ups where nodes can join the network after its initial deployment. The simulation results demonstrate the performance of the algorithm when parameters such network size, network density and percentage of initially active nodes vary. The algorithm is performed well when the parameter controlling the collision handling mechanism is set properly.

Fault-tolerance for distributed detection is introduced in Chapter 4. Current approaches are described and a new localized algorithm is proposed to account for cases where nodes in the same network have different accuracy levels. The performance of the new algorithm is studied analytically and using simulation. In this chapter, we also introduce methods for nodes to dynamically learn their reliability levels used to update node weights in the detection algorithm. These methods are based on the moving average and the geometric moving average filters. Two new detection error

models are introduced. The first model takes into account neighbor distance when assigning weights to different neighbors. The second model accounts for the importance of the geographical distribution of the voting quorums.

Chapter 5 concludes this thesis and introduces future work.

CHAPTER II

SENSOR NETWORKS SIMULATION

This chapter presents the *Georgia Tech Sensor Network Simulator* (*GTSNetS*), a new sensor network simulation environment that enables the development and evaluation of algorithms for large-scale sensor networks. It allows users to evaluate the effects of different architectural choices and strategies on the lifetime and performance of a particular sensor network. The new simulator tool can also be used to evaluate new approaches such as new sensor network algorithms and network protocols.

Section 2.1 motivates the need for the new simulator. Existing sensor network simulators are discussed and contrasted with *GTSNetS* in Section 2.2. Section 2.3 discusses the design of *GTSNetS* and presents in detail the implementation of the simulator. Section 2.4 presents an extension of *GTSNetS* to allow the simulation of networked control systems. Section 2.5 presents some results demonstrating some of the simulator capabilities. Section 2.6 concludes the chapter.

2.1 Motivation

A sensor node can be divided into three main functional units: a sensing unit, a communication unit and a computing unit. Since a sensor node is typically powered by a battery, energy savings are of critical importance for a sensor network. In fact, these networks are often deployed in unfriendly remote environments such as a forest. This isolation makes changing the battery or charging it expensive, if not impossible. For this reason, maximizing the lifetime of a sensor network, while keeping an acceptable performance level, has seen substantial interest from the research community. The lifetime is maximized through innovative approaches such as new network protocols consuming less energy, smarter collaboration strategies between neighboring nodes

and more efficient processor power management, just to name a few. Comparison between different available architecture alternatives is also necessary when designing a sensor network for a special application. In fact, the designer might need to choose among different collaboration strategies, different network protocols, or decide on other design parameters, such as the node density of the network.

It is usually not possible to deploy sensor networks of realistic sizes to test and validate these new approaches. In fact, sensor networks might consist of several thousands if not hundreds of thousands of elements, which makes it costly and time consuming to deploy such a large network for testing. An alternative approach would be to use theoretical modeling and analysis to compare different architecture alternatives. However, these models often fail to capture many important aspects of sensor networks such as energy consumption and complex interactions between individual nodes in a large network. For these reasons, simulation becomes the only viable alternative to validate new design approaches for sensor networks.

This chapter presents the *Georgia Tech Sensor Network Simulator* (*GTSNetS*), a new simulation tool for wireless sensor networks. It is built on top of the *Georgia Tech Network Simulator* (*GTNetS*) [87, 88], which is an event-driven object-oriented network simulation tool. *GTSNetS* inherits and extends all the design decisions of the existing *GTNetS*, which makes it suitable for simulation of large-scale sensor networks. To the best of our knowledge, *GTSNetS* is the only sensor network simulator capable of handling networks of several hundred thousand nodes. This level of scalability is important as recent years have seen a steady increase in the size of deployed sensor networks. The size of these networks grew from few tens of nodes in 2002 to few hundreds in 2003 and over a thousand in 2004 for the ExScale project [9]. The ExScale border monitoring project has a final target deployment of $10,000$ nodes. As the sensor network field matures, sensor networks of tens and even hundreds of thousands of nodes are envisioned for industrial, military, agricultural and medical

applications to name a few [9].

The features of *GTSNetS* compared to existing simulators, include:

1. A unifying framework of existing energy models for the different components of a sensor network. This allows the user to choose the energy model that best suits his simulation scenario.

2. *GTSNetS* provides several models for the accuracy of sensed data and allows the addition of new models. Modeling the accuracy of sensed data helps, in particular, in the understanding of the trade-off between the different performance metrics such as between the quality of the measurements and the network lifetime.

3. This simulator allows the user to choose among different implemented alternatives: different network protocols, different types of applications, different sensors, different energy and accuracy models. New models, if needed, can be easily added. This makes *GTSNetS* suitable for simulating sensor networks since such networks are application-dependent and their diversity cannot be represented in a single model. Such modularity allows an independence between the different modules, which makes it easy for the user to extend the simulator by adding new modules (inherited from the existing ones) or modifying existing modules without affecting other parts of simulator.

4. *GTSNetS* can be used to simulate networked control systems consisting of a set of nodes having sensing, control and actuation capabilities.

5. Finally, *GTSNetS* can be used to collect detailed statistics about a specific sensor network at the functional unit level, the node level as well as at the network level. The user has extended control over the type and amount of tracing data to collect. This control can be leveraged for a trade-off between memory and

computing resource requirements (for tracing) and the size and complexity of the simulated network.

GTSNetS is distributed under the GNU General Public License and is available at: `http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/download.htm`.

2.2 Large-scale sensor network simulation

Several sensor network simulators are available and are in widespread use by the research community. However, most of these simulators lack models for important aspects of sensor network simulation such as energy consumption models and sensor accuracy models. In addition, most of these simulators do not scale well with the size of the network and can simulate only networks that are of the order of few tens of thousands of nodes at best.

SensorSim [73] was built upon the widely used *ns-2* [63] simulator. It provides models for different parts of a sensor network architecture: battery, sensors, radio, CPU, etc. It also models the power consumption of these components. However, the provided power models are somewhat simplistic. It is assumed that each task consumes a fixed amount of energy. *GTSNetS* on the other hand includes several energy models for each of the node functional units (sensing, communication and computing) allowing the user to choose the most appropriate model for his scenario and to easily add new models when needed. A major feature of SensorSim is its ability to support hybrid simulation: integration between the simulator nodes and real sensor nodes collecting real data and running real applications. However, SensorSim does not scale very well with the network size, and to the best of our knowledge has not been used for networks of more than $1,000$ nodes. In addition, SensorSim is no longer maintained and not publicly available.

sQualnet [106] follows the same approach as SensorSim and offers similar features using Qualnet [90] as a base simulator instead of *ns-2* as is the case in SensorSim.

The main advantage of sQualnet over SensorSim is its higher scalability. However, to the best of our knowledge, it has not been demonstrated to support networks of more than $10,000$ nodes. A main limitation of sQualnet lies in the fact that it was designed specifically for applications based on the SOS operating system and does not generalize to all types of sensor network applications.

SENS [101] is another sensor network simulator. A main feature of SENS is its detailed environment model. However, SENS lacks realistic energy models in that it assumes constant power consumption values in each operational mode. Like *GTSNetS*, SENS is implemented in C++ and promotes multiple implementations of various architectural choices in particular network modules. However, *GTSNetS* offers more choices such as multiple energy models, accuracy models and application scenarios which are lacking in SENS. In addition, to the best of our knowledge, SENS cannot handle networks of more than $10,000$ nodes.

J-Sim [98] and SENSE [21] attempt to handle the scalability problem by using a component based approach rather than an object-oriented approach. J-Sim is shown to simulate networks of several hundred nodes using less memory than *ns-2*. However, the use of Java introduces a new set of inefficiencies and the inter-communication model incurs substantial overhead. SENSE uses C++ instead of Java to improve scalability. SENSE is proven to simulate networks of similar sizes as in J-Sim and *ns-2* while using less memory resources. The scalability of SENSE in terms of network size ranges from $5,000$ to 500 nodes depending on the communication patterns in the network. OMNet++ [60] is another component based simulator. It has been shown to simulate sensor networks of several thousand nodes.

The SWAN [54] simulator is shown to be capable of simulating sensor networks on the order of ten thousand nodes. However, SWAN focuses more on the wireless ad-hoc network aspect of sensor networks and less on the sensing function. To the best of our knowledge, it has not been used to simulate sensor networks of more than ten thousand

nodes. In addition, SWAN does not take into account the energy consumption, which is necessary to model considering the importance of power consideration in sensor networks.

TOSSF [78] is an adaptation of SWAN, developed by L. F. Perrone and D. M. Nicol of ISTS, which simulates the execution of TinyOS applications at the source-level. It relaxes a major restriction of TOSSIM [52], the other TinyOS applications simulator. Unlike TOSSIM, TOSSF does not require that all nodes within the simulated network run the same set of applications. However, these two simulators cannot be used to validate new applications for general use on sensor networks running any platform, since they are specifically tailored for TinyOS. In addition, these two simulators focus on the application behavior and do not necessarily capture all the aspects of a sensor network. Other AVR microcontroller simulators such as Avrora [104] and ATEmu [82] are cycle accurate simulators that can simulate any application by executing the code at machine-level. These instruction-level simulators have the advantage of higher accuracy but suffer from lack of scalability. ATEmu for instance can simulate only networks of up to 120 nodes, while Avrora uses a less accurate synchronization mechanism to improve scalability. It is shown to simulate networks of up to $1,750$ nodes on a dual core processor and $10,000$ nodes on a Sun Enterprise system. In contrast, *GTSNetS* sacrifices the accuracy of instruction-level simulation to scale to networks of up to $200,000$ nodes on a single core workstation.

In addition to sensor network specific simulators, classical network simulators such as *ns-2* [63], Glomosim [114], Qualnet [90] and OpNet [68] can be used to simulate certain aspects of sensor networks. However, these simulators do not model many important features of sensor networks such as sensed object, realistic energy models, accuracy models and sensor network specific communication protocols.

The simulation of networked control systems has received a growing amount of interest lately. In particular, two main simulation frameworks for networked control

systems that have been proposed. These are: TrueTime, a MATLAB-based simulation framework and an Agent/Plant extension to *ns-2*.

TrueTime [41] is based on MATLAB/Simulink and allows the simulation of the temporal behavior of multi-tasking real-time kernels containing controller tasks. It proposes two event-driven Simulink blocks: a computer block and a network block. The computer block is used to simulate computer control activities, including task execution and scheduling for user-defined threads and interrupt handlers. The network block is used to simulate the dynamics of a computer network using parameters such as message structure and message prioritizing function. However, this network block is not general enough to simulate various types of networks, especially sensor networks. It also suffers scalability problems.

The *ns-2* simulator was extended to allow simulation of the transmissions of plants and controllers in a networked control systems [16]. The authors added a plant and an agent classes to *ns-2*. To the best of our knowledge, this solution is not yet interfaced with any of the *ns-2*-based sensor network simulators [73]. In addition, it does not allow the simulation of large networked control systems.

2.3 GTSNetS *Design and Implementation*

To help understand the situations for which the simulator can be used, a typical sensor network scenario is presented here. This example consists of a network of $1,000$ sensor nodes and a sink node in an area of 500 meters by 500 meters. The nodes are placed randomly within the region. Every sensor node has a temperature sensor. An energy model is chosen for each of the different units: sensing, communication, and computing units. Directed diffusion is used as the routing protocol.

The sink node receives temperature readings every 30 seconds from nodes in the region defined by the lower left corner (25, 25) and the upper right corner (200, 200) for a period of 6 hours. It sends a request throughout the network specifying the

desired information. Every sensor node that has a temperature sensor receiving this request will check if it is in the appropriate region. If so, it will activate its sensor and collect the temperature measurement every 30 seconds. Every time the sensed data is collected, the sensing energy expended is incremented as well as the total energy consumption of the node. The battery then updates its remaining energy accordingly, and the node lifetime is updated.

The sensed data is then given to the application for processing and transmission to the base station. The response is sent back to the base station through the communication unit (interface and protocol stack). After every task the appropriate energies as well as the lifetime are updated.

Intermediate nodes receiving and forwarding the response will also update their communication energy and the battery remaining energy. If packet tracing is enabled, the message is traced to a log file whenever a packet receipt or transmission occurs.

The following subsections describe how the different functionalities and models needed to simulate this scenario and other sensor networks scenarios are designed and implemented in *GTSNetS*.

2.3.1 Design Logic

GTSNetS is built as an extension of the *Georgia Tech Network Simulator* (*GTNetS*) [87, 88] as a simulation framework for sensor networks. As *GTNetS* is written entirely in the C++ language using an object-oriented methodology, it was possible to leverage many of the existing functionalities of *GTNetS* with moderate effort.

The *GTNetS* is a full featured general-purpose wired and wireless networks simulation environment. It is designed with the objective of closely mimicking the way real networks are structured while handling the simulation of networks. Scalability is achieved through careful resource optimization (e.g., reduction of the number of outstanding events, memory management and optimization and reduction of log files

size). There is a clear separation between the protocol stack layers as in standard network protocols. *GTNetS* has models for the various components of a network including nodes, network interfaces protocol layers and many popular network protocols. It also provides models for end-to-end user applications. Packets are modeled as composed of a list of protocol data units (PDUs). PDUs are appended and removed from the packet as it moves down and up the protocol stack. *GTSNetS* builds on top of *GTNetS* by providing models to enable sensor network simulation. New models include:

1. Models for each of the different functional units composing a sensor node: sensing unit, computing unit and communication unit.

2. Sensing accuracy models.

3. Battery model.

4. Several energy models for each functional unit.

5. Sensor network-specific protocols and applications.

6. New tracing capabilities to enable the collection of statistics on energy consumption at network, node and functional unit levels.

7. Models to enable the simulation of networked control systems where a sensor network is used in the context of a distributed control application.

These different new models and their implementation are discussed in detail in the next subsection. Figure 1 gives a schema of the most important new classes in *GTSNetS* , and the existing *GTNetS* base classes used to support the sensor network simulation environment. It also illustrates the relationships between the different classes.

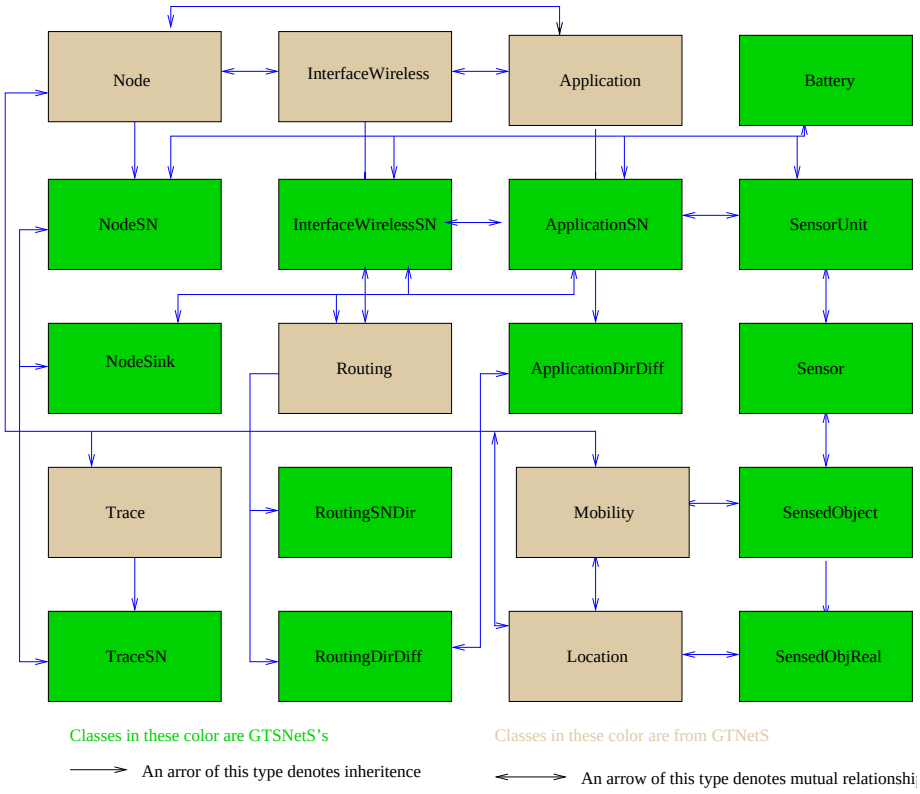

Figure 1: GTNetS and GTSNetS classes and their relationships

GTSNetS inherits and benefits from the basic design philosophy of the existing *GTNetS* as discussed in [87, 88]. *GTSNetS* is built in a modular and efficient manner, leading to the capability to simulate large-scale wireless sensor networks along with adaptability, extensibility and ease of use.

GTSNetS is designed in such a way that it would not impose any architectural or design decisions on the user who wants to simulate a particular sensor network. Yet, *GTSNetS* is not a "do it yourself" type of simulator. Rather, the user can choose from various implemented alternatives: different energy models, accuracy models, network protocols, applications and tracing options. Several different methods for each of these choices are included in the baseline implementation. This adaptability characteristic of *GTSNetS* can be used for example to compare different design choices (e.g., different communication protocols) or to implement a hierarchical sensor network with different nodes having different functionalities by attaching different applications to different nodes. A similar approach can be used to implement a heterogeneous sensor networks with different sensor nodes having different types of sensing capabilities for example.

Should the implemented models not be sufficient for a particular simulation scenario, the user can extend easily the simulator by adding new models or modifying the existing ones. The ease of extensibility comes from the object-oriented C++ implementation of the simulator, with many base classes that can be easily inherited.

This modularity can also be used to tradeoff accuracy for scalability. In particular, if a simulation scenario is intended to test a communication protocol and is not concerned with the sensing functionality, then it is not necessary to use a realistic implementation of the sensing unit (and corresponding sensors, accuracy models and sensing objects). For example, instead of having a different sensing unit attached to each of the N nodes, a single sensing unit can be attached to all nodes. This can result in substantial simulation memory requirements not only because of the $N-1$ less sensing units and sensing objects to store, but also of the reduction in number of

sensing events.

The scalability is also ensured by using model-based simulation rather than instruction-level simulation, while maintaining an acceptable level of realism. In addition, *GTSNetS* is implemented in an efficient and modular manner taking into account the nature of sensor networks. For example, in these networks communication is performed via wireless devices and the network topology can be dynamic with packets transmitted typically in a localized broadcast. By eliminating the need for nodes to maintain and update routing tables for most sensor network protocols, *GTSNetS* reduces the memory and processing requirements. In addition, *GTSNetS* extends *GTNetS* approach of fine-grained control of the logging of simulation events. By implementing many tracing options (such energy states, message exchange, node wake-up times, etc.) and allowing the user to choose exactly which ones to log and discard the others, *GTSNetS* allows for a trade-off between memory requirements and the amount of simulation events logging.

Because of the importance of lifetime in sensor networks, *GTSNetS* provides the capability to track the overall lifetime of a simulated network. The network lifetime here is defined as the amount of time during which the network has been able to accomplish its tasks according to the quality of service requirements fixed by the user. For example, if the network is asked to collect and relay to the base station a specific type of sensed data with the requirement that an update is received by the base station at least twice every T seconds along two different paths, then the network is considered dead the first time two updates are not received for a period of T seconds. In this example, the sensor network specified task is to collect and relay the specified type of data. The quality of service requirement, here, consists of the duplicate paths. This definition is different from the classical definition that considers the network dead when the first node runs out of energy. It is more adapted to sensor networks where redundancy is very common. The network can continue functioning

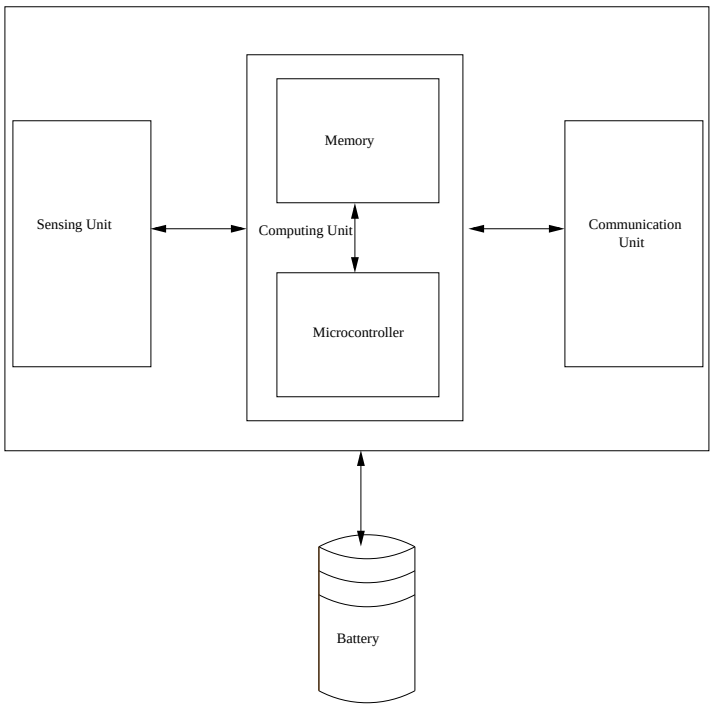

Figure 2: Sensor node functional architecture

with a good performance even after the death of a significant percentage of its nodes. The simulator also measures the energy consumption of each one of the functional units and provides detailed statistics for each, allowing the user to study the effect of different architectural choices on lifetime and energy consumption.

GTSNetS inherits mobility support from the existing *GTNetS* and thereby, allows the specification of mobile sensor nodes, moving sensed objects, as well as a mobile base station.

2.3.2 Implementation

A sensor node is composed of three main functional units: a sensing unit, a communication unit and a computing unit. A battery provides energy for these three units. These different elements shown in Figure 2 are described in more detail in the next subsections.

In addition to regular sensor nodes, a sensor network can contain one or more sink

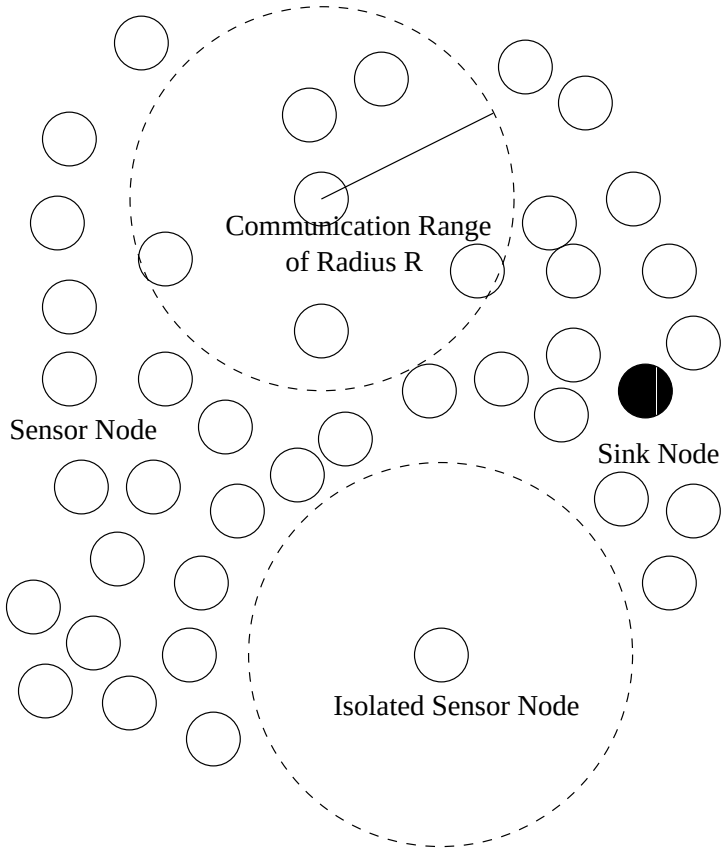

Figure 3: Small sensor network

nodes (base stations). These sink nodes interact with sensor nodes to collect sensed data and serve as a relay to the outside world. A sink node has a similar architecture to that of a regular node. The main difference is that a sink node does not have a sensing unit. A sink node is also considered, by default, to have an unlimited energy source, and therefore, there is no need to model a battery to characterize its energy consumption. However, with minor changes, the user could choose to attach a battery to the base station in the same way as for a regular sensor node. Figure 3 illustrates a small sensor network, where each node has a limited communication range and becomes isolated when no other node is within its communication range.

GTSNetS provides models for all the elements of a wireless sensor node: computing unit (`class NodeSN`), sensing unit (`class SensUnit`), communication unit (`class`

`InterfaceWirelessSN`) and battery (`class Battery`). The three units consuming energy (Sensing unit, Communication unit and Computing unit) each have several energy models to characterize the energy consumption within this unit. The user can choose from these models or add his own energy models.

These different elements and their implementations are described in more detail in the following subsections.

2.3.2.1 Sensing Unit

The sensing unit (`class SensorUnit`) consists of one or more sensors (`class Sensor`) attached to a particular node. Each sensor is defined by the type of data it can collect, such as a temperature sensor, IR sensor, camera, etc.

Each sensor senses one or more objects. A sensed object can either be a physical object (`class SensedObjReal`) such as a moving target, or it can be the environment of the sensor node (`class SensedObject`), which might be the case when collecting temperature. When sensing a real object, meaningful data can be collected only if the sensed object is within the sensing range of the specific sensor. The sensor can be in one of two states: active or inactive. The change between these two states is controlled by the application (running on the computing unit). When active, the sensor is constantly collecting data at a fixed sensing rate.

The simulator provides accuracy models to to account for measurement errors. The accuracy level of a sensor depends on several factors including the sensor characteristics, the sensing environment and the sensed object. The user can choose from three different accuracy models that are available in the simulator or easily implement additional ones if necessary.

The first accuracy model consists of a simple random variation of the sensed data around the real value within a fixed percentage depending on the particular type of sensor and the sensing environment. If the real value is S_r and the fixed percentage

is γ, then the supplied value is given by S_d:

$$S_d = (1 + \frac{\epsilon}{100})S_r \tag{1}$$

where ϵ is a random value between $-\gamma$ and $+\gamma$. This relationship is only valid when the sensed object is within the sensing range of the sensor. This model can, for example, be used for a sensor collecting data on the characteristics of its environment, such as a temperature sensor. In this case, the distance is not relevant.

The second model is a distance-based accuracy model based on the observation that sensors are more accurate when the sensed object is closer. In this case, the error percentage, γ, is no longer fixed. It is a function of the distance of the sensor node to the sensed object, with closer distances leading to smaller errors. If the distance between the node and the sensed object is given by d, then γ is given by:

$$\gamma = k \times d^{\alpha} \tag{2}$$

where k is a constant. This relationship can be linear ($\alpha = 1$), or exponential, i.e. the accuracy of the sensed value decreases exponentially with distance. As in the first model, this relationship is only valid if the sensed object is within the sensing range of the sensor, i.e. $d < R_s$. The value supplied by the sensor to the computing unit is given by a similar relationship to the one in the first model, but using the new distance based value of γ. For an example use of this model, consider the case of a sonar tracking a moving target. The information obtained by this sensor becomes more and more accurate as the target gets closer.

In a similar way, the third sensing accuracy model is based on the observation that in certain applications larger sensed objects are more likely to be detected than smaller objects. In this case, the error percentage, γ, is a function of the size of the sensed object, with larger objects leading to smaller errors. If the size of the sensed object is si, then γ is given by:

$$\gamma = \frac{k}{si^{\beta}} \tag{3}$$

where k and β are constants. Again, this relationship is only valid if the sensed object is within the sensing range of the sensor, i.e. $d < R_s$.

The fourth model considers an additive normally distributed error term ϵ with mean 0 and variance σ^2. The parameter σ is supplied by the user and is a function of the sensing environment and the sensor characteristics. The sensed data as obtained by the sensor is given by:

$$S_d = S_r + \epsilon \tag{4}$$

This model can be used, for example, to model the accuracy in the case of an event region detection. In such case, individual nodes are interested in determining whether they are located in a region where a specific event is happening. By fusing the data from all the sensor nodes, the base station can determine the boundaries of the event region. A node determines whether it is in the event region by looking at the value of its sensed data. It is assumed, for simplification, that a low sensor reading indicates the absence event, while a high reading indicates the presence of an event. Assume that the real sensed value in the absence of event has a mean of m_f, while in the presence of an event the mean value is m_e. By considering a normal distribution of the error term ϵ with a mean of 0 and a variance of σ^2, it is possible to compute the probability of event detection failure. That is the probability of a node declaring itself in an event region when it is not or the probability of the node declaring that it is not in an event region when an event is occurring in its sensing range. This probability p is given by:

$$\begin{aligned} p &= P(S_i = 0|T_i = 1) = P(S_i = 1|T_i = 0) \\ p &= Q(\frac{m_f - m_n}{2\sigma}) \end{aligned} \tag{5}$$

Here, Q is the tail probability function of the Gaussian distribution. The binary variables T_i and S_i represent, respectively, the actual state (presence or absence of event) and the state as perceived by the node from it sensor reading, which can be

erroneous. The determination of the value of p is important and can be used to derive an appropriate fault-tolerance mechanism as discussed in Chapter 4

The sensor node sends the confidence level along with the corresponding sensed data to the base station. The confidence level is an indication of the maximum difference between the value measured by the node and the real value. The confidence levels can be used, for example, to weight the sensed values coming from different nodes when they are fused at the base station.

The simulator keeps a record of the cumulative amount of energy consumed by the sensing units. Sensors can be classified as passive or active. Passive sensors, such as temperature and seismic sensors, consume a negligible amount of energy compared to the consumption of computing and communication. On the other hand, active sensors, such as ultrasonic and radar, can consume a significant amount of energy. Several sources of power consumption can be identified at the sensing unit level [85]. These include: signal sampling and conversion of physical signals to electrical ones, signal conditioning and analog to digital conversion before sending to the processing unit.

Three different energy models, which characterize the sensing energy consumption are implemented in the simulator. These implemented energy models vary in terms of complexity, accuracy and level of details. The user can choose among these three models or add his own energy model if necessary.

The first energy model is a simple model, which assumes a linear relationship between the sensing energy and the size, in bits, of the sensed data. It considers that nodes collect data at a fixed rate, b bits/second. The energy consumed by the sensing unit to collect S bits is then:

$$E_{sense} = E_s S \tag{6}$$

where E_s is the energy needed to sense a bit, assumed to be constant, typically $E_s = 50$ nJ/bit [11, 29]. This model can be used in the case of a sensor that has a

fixed sensing range, in meters, that cannot be adjusted for changing tasks or saving energy.

The second energy model takes into account the effect of the sensing range. The sensing energy, E_{sens}, is considered to be exponentially proportional to the sensing range R. The sensing power is given by:

$$E_{sense} = E_0 R^a \tag{7}$$

The parameter E_0 is a function of the size of the sensed data, considered constant here, and the sensor characteristics. The value of the exponent a depends on deployment factors and terrain characteristics [75]. This model can be used to appropriately adjust the sensing range to minimize energy consumption while maintaining an appropriate sensing range. This could be done in the case of target tracking for instance.

The last model breaks down the sensing energy into two parts: energy consumed by the amplifiers and the energy consumed by the analog to digital conversion process. This model allows for a better trade-off between energy and performance. This model is described in more detail in [27].

2.3.2.2 Communication Unit

The communication unit is in charge of relaying sensed data to the sink node and other sensor nodes when needed. It is composed of the wireless interface and the protocol stack.

The wireless interface (`class InterfaceWirelessSN`) is based on the wireless interface model in *GTNetS*, extended to track the energy consumption of the interface using one of several parameterized energy models. This energy is broken down in terms of transmission and reception energies. The overall communication energy is updated every time a reception or a transmission occurs.

The energy consumption of the communication unit depends on several factors.

These include the modulation scheme, the data rate, the transmission distance and the operational mode. A communication unit can operate in several modes. The number and composition of these modes can vary from platform to platform, an example can be: active, idle, and sleep. The last two modes imply constant power consumption. A constant power is also consumed during a change of operational mode. This power requirement is generally quite small, and *GTSNetS* does currently not have a model for this type of energy consumption. However, if nodes change often between operational modes, the energy consumption for changing between modes could become an important percentage of the overall energy consumption and should be considered in the overall lifetime computation [85].

The communication energy consumption can be described in terms of transmit and receive consumptions. The transmit energy consumption depends on the transmission range and the energy consumed in the transmission circuitry. Furthermore, both transmit and receive energy consumptions depend on the message size.

A simple communication energy model considers a linear relationship between the transmission energy and the message size in bits and an exponential relationship with the transmission range. For the reception energy, it is assumed to depend only on the message size and has a linear relationship with the message size. The energy consumption when transmitting or receiving r bits between two nodes n_1 and n_2 with a distance of $d(n_1, n_2)$ is given by:

$$\begin{aligned} E_{tx} &= (\alpha_{11} + \alpha_2 d(n_1, n_2)^n) r \\ E_{rx} &= \alpha_{12} r \end{aligned} \tag{8}$$

where E_{tx} and E_{rx} are the transmission and reception energy consumption, respectively. The parameters α_{11}, α_2 and α_{12} are constants. Typical values are $\alpha_{11} = 45$ nJ/bit, $\alpha_{12} = 135$ nJ/bit, $\alpha_2 = 10$pJ/bit/m2 (for $n = 2$) and 0.001 pJ/bit/m4 ($n = 4$) [11].

Another model is to compute the minimum power required to counter the thermal noise [27]. The thermal noise TN is given by:

$$\mathrm{TN} = kTB \tag{9}$$

where k is Boltzmann's constant, T the temperature (in Kelvin) and B the noise bandwidth (assumed equal to the symbol rate). In the previous equation, the required transmission power P_t to counter the thermal noise is given by:

$$P_t = kTB \tag{10}$$

The signal power loss during a packet transmission is proportional to the square of the ratio of distance to wavelengths between the two nodes. The receiver signal power must be sufficient to properly detect the received message. The receiver signal strength also depends on the antenna gain, G_{ant}, at the receiver. The receiving power P_r is given by:

$$P_r = \frac{P_t G_{ant}}{16\pi^2(\frac{d}{\lambda})^2} \tag{11}$$

where d is the distance between the two nodes and λ is the wavelength.

For the networking aspect, *GTSNetS* can use the normal TCP/IP protocol stack (inherited from *GTNetS*), although simpler communication paradigms may be ultimately used for sensor networks. Indeed, a non-layered protocol stack would be likely simpler in terms of computation, and thus would be more energy efficient.

Packet routing can be problematic for sensor networks since most of the routing protocols for ad-hoc networks, such as DSR [45] and AODV [76], are unsuitable for sensor networks due to the overhead of route discovery and high failure rates in sensor networks. In addition, many sensor networks do not have any kind of global identification. Rather, sensor nodes are aware only of their local neighbors.

For these reasons, several routing protocols for sensor networks are implemented in the simulator. In the first protocol (`class RoutingSNDir`), sensor nodes are assumed

to have a communication range large enough to communicate sensed data directly to the sink station (base station). Therefore, there is no need for route computation. This protocol is implemented for benchmark purposes and is not expected to be used in any application of large-scale sensor networks.

The second routing protocol implemented in *GTSNetS* is directed diffusion presented in [44]. In directed diffusion, requests are addressed by the type of sensed data (such as temperature, seismic, etc) and the region of interest. The sink node floods the network with an interest (request) for a certain type of data coming from a certain region. It also attaches to the interest the duration of validity and how often it needs the data. A sensor node receiving this request will check if it has a sensor that can provide the requested information, and if it is in the region of interest. In such a case, it activates its sensor and starts collecting and sending data. Otherwise, it broadcasts the request to its neighbors. It also registers the interest along with the neighbor or neighbors (gradients) from which it was received. Sensed data is sent it to these neighbors rather than addressing it to the sink node. When a node receives sensed data from one of its neighbors, it checks if it has previously registered an interest for this particular type of data. If so, it forwards the response to all the neighbors for which it has gradients. The sink node can choose to reinforce some of these paths by forwarding the interest only along the appropriate path and with a higher frequency (how often data is needed). To avoid broadcast storms during the flooding phase, sensor nodes will only forward requests that were not forwarded previously.

Several geographical routing protocols are also implemented in the simulator. For these geographical routing protocols, nodes are assumed to be location-aware either through GPS or other sensor network localization schemes, such as the one in [89]. The first geographical routing protocol allows the collection of data from sensor nodes

by the base station in an energy efficient manner. It does not allow multi-hop communication between two sensing nodes, it only handles messages between a sensing and node and the sink node. Routing of messages sent from sensor nodes to the base station use nodes location-awareness to deliver messages in an energy efficient way. It is assumed that the location of the base station is known to all nodes in the network. This can be achieved through an initialization phase where the base station floods the network with a message containing its location. In this case, nodes do not have to include the destination (base station) location in the header, which reduces the communication energy consumption.

Several approaches to deciding which neighbors route a specific message are implemented in *GTSNetS*. In one approach, all neighboring nodes that are closer than the current source (initial source or current forwarder) forward the message if it has not been received previously. This approach allows for a certain degree of redundancy and fault-tolerance, since most messages will be delivered by several neighbors. A second approach reduces the energy consumption at the expense of fault-tolerance by having only the closest neighbor to the base station forward the message. In both cases, only nodes that are between the current source and the base station forward the message.

A second geographical routing protocol is implemented to allow for the communication between any two nodes in the network. These nodes can both be regular sensor nodes or a sensor node and the base station. This protocol uses similar approaches to make routing decisions as in the previous one. The main difference is that the location of the final destination has to be included in the message, since this destination is not always the base station. This protocol can be used in cases where collaborative sensor network algorithm require communication between sensor networks that are not within each other neighborhood.

It is planned to add other routing protocols such as the geographical routing

protocol in [112] and the directed flood routing protocol in [62].

For MAC protocols, both IEEE 802.11 [43] and Bluetooth [95] are inherited from the basic *GTNetS*. The reader is referred to [86] for a detailed description of the 802.11 protocol implementation in *GTNetS*. A detailed description of the *GTNetS* Bluetooth implementation can be found in [115].

For MAC protocols, both IEEE 802.11 and Bluetooth are inherited from the basic *GTNetS*. The IEEE 802.11 implementation in *GTNetS* corresponds to the specification in [43]. It uses Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) with acknowledgments for channel access. RTS/CTS messages are exchanges before sending unicast messages. The reader is referred to [86] for a detailed description of the 802.11 protocol implementation in *GTNetS* and its comparison with the implementations in other popular simulators. The Bluetooth implementation corresponds to the Bluetooth specification version 1.1 [95]. It models the detailed behavior of the lower layers of the Bluetooth protocol stack, including Baseband, LMP, L2CAP, and BNEP. A detailed description of the *GTNetS* Bluetooth implementation can be found in [115].

It must be noted here that it is easy to extend *GTSNetS* to implement new MAC protocols or individual functionalities of MAC protocols (e.g., collision avoidance). This is particularly useful for sensor networks where it is sometimes not possible to assume that a specific MAC protocol is established. For example, during the initialization of a sensor network no communication structure or MAC protocol can be assumed [50]. In this case, initialization algorithms can incorporate simple functionalities traditionally provided by a MAC protocol. As an example, we implemented in *GTSNetS* a simple collision avoidance and handling mechanism for the ID assignment algorithm proposed in Chapter 3. The collision is handled through avoidance and recovery. Collision avoidance is implemented using a simple mechanism where each node waits a random time before sending a message. The value of this random time

is chosen between a minimum and a maximum value. The interval of possible values ($maximum - minimum$) is extended when the collision rate increases and reduced when the number of collisions decreases. Recovery from collisions is implemented through message retransmission when no acknowledgment is received.

2.3.2.3 Computing Unit

The computing unit consists of a micro-controller unit (MCU) running a set of applications. It controls the sensing unit, performs the signal processing and executes the communication protocol. An application (`class ApplicationSN`) is attached to each sensor node. This application can request a sensor to activate or deactivate itself and can change the rate at which sensed data is collected. Another role of the application is to model any necessary processing on sensed data before sending it to other nodes in the network. It also receives and processes messages from other nodes. These messages can contain sensed data, for example when neighboring nodes aggregate their sensed data before sending a common message to the base station. Nodes can exchange other types of messages such as messages related to collaborative reconfiguration, self-initialization or cluster formation and cluster-head election.

Different applications can be implemented by extending the basic sensor network application class (`class ApplicationSN`). Currently, several applications are implemented including a distributed identification application, several fault-tolerance applications and several reconfiguration application.

The distributed identification application is used to assign globally unique IDs to all nodes in the sensor networks. This is done in an energy efficient way by using only the minimum number of bytes required to code the number of IDs corresponding to the total number of nodes in the network. The algorithm executes in a distributed during the initialization phase. A detailed description of this application is given in [69].

Two applications were implemented to handle the fault-tolerance problem in sensor networks by extending `ApplicationSN`. The first approach implements a Bayesian fault-tolerance algorithm presented first in [49]. The algorithm allows each node to detect the presence of a sensor failure by comparing its sensor reading with the ones of its neighbors using a spatial correlation mechanism. A problem with this approach is that it makes two unrealistic assumptions consisting of assuming that all nodes have identical failure probabilities that are known prior to the deployments. The simulation results obtained using *GTSNetS* agree with the results presented in the original paper that were obtained using MATLAB and with the theoretical results. This increases the confidence in the correctness of the models used in *GTSNetS*.

The Bayesian fault-tolerance algorithm was extended to relax the assumptions of identical and known failure probabilities. This extended solution integrates the possibilities of nodes having different failure probabilities, which could come, for example, from changing energy constraints of some of the nodes or nodes having sensors with different capabilities. This solution does not require the failure probability to be known prior to the deployment. Rather, each node learns its failure probability as it operates by comparing its performance with the performance of its neighbors. For more details on this application and on its performance compared to the original Bayesian algorithm, refer to [70].

Several applications were also implemented to handle the reconfiguration problem in sensor networks. The problem consists of maximizing the overall network lifetime while maintaining a required quality of service. This is done by turning some of the nodes off when they are not needed to maintain the required quality of service. The required quality of service could be defined, for example, in terms of coverage and connectivity levels. To solve the reconfiguration problem, it was decided to interface *GTSNetS* with GLPK (GNU Linear Programming Kit). GLPK is a software package intended to solve large-scale optimization problems, such as linear programming (LP)

and mixed integer programming (MIP)[59].

The energy consumption of the MCU depends greatly on the required level of performance, and therefore, will be determined by the choice of MCU offering the level of performance required by the specific sensor network application. MCU can run in several modes (active, idle, sleep) and uses energy during transition between modes that needs to be included in determining the overall energy consumption of the unit. Two computing energy models are implemented in *GTSNetS*. The first computing energy model considers that there is a constant energy dissipation per bit processed (E_c). The computing energy is, then, given by:

$$E_{Comp} = E_c b \tag{12}$$

where b is the number of bits in the processed stream. E_c can range from 1 pJ/bit to tens of nJ/bit depending on the application [11].

The second model considers that the energy consumption in the computing unit can be divided into the switching energy E_{Switch} and the leakage energy $E_{Leakage}$ [7, 37, 94]. The switching energy is consumed when transferring between internal states of the micro-controller, while the leakage energy represents the energy lost in the idle mode. The leakage energy can be neglected in some cases, but it can also count for up to 10% of the computing energy in other cases depending on the type of micro-controllers.

$$\begin{aligned} E_{Switch} &= C_{Total} V_{DD} \\ E_{Leakage} &= V_{DD} \frac{I_0 e^{\frac{V_{DD}}{nV_t}} N}{f} \end{aligned} \tag{13}$$

C_{Total} is the total capacitance switched by the computation, V_{DD} the supplied voltage, V_t the thermal voltage, N the number of cycles and f the frequency of the processor. The parameters n and I_0 are processor dependent [37].

2.3.2.4 Battery

The battery (`class Battery`) is modeled as an energy reservoir initially with a fixed amount of available Joules. The remaining energy of the battery is updated (reduced) every time an activity that consumes energy occurs: sensing, processing, transmitting, or receiving. Every time the remaining energy is updated, the node lifetime is also updated by adding the time since last update. The node lifetime is measured as the time since the node starts: the node creation in the simulator or the node first activation. The node dies when this remaining energy reaches a minimum threshold specified by the user.

Modeling the battery as a reservoir of joules is rather a simplistic approach. A more realistic model that accounts for the non-linear and recovery effects are planned for addition.

2.3.2.5 Sink Node

The base station has all the components of a regular sensor node except the sensing unit and the battery. It is in charge of gathering the sensed data from sensor nodes. It also keeps track of the network lifetime. The network lifetime here is defined as the amount of time during which the network has been able to accomplish its tasks (for example collecting and relaying to the base station certain type of sensed data) according to quality of service specification fixed by the user (for example, report this data every T seconds along two different paths).

2.3.2.6 Tracing

GTSNetS provides for extensive packet tracing by extending the existing tracing capabilities of *GTNetS*. By default, tracing occurs every time a message is sent or received by a sensor node, or by the sink node. The user is given a wide set of tracing options: the different types of energies (example sensing) at node level as well as network-wide, node and network lifetime, the sensed data, location and whether the

node is dead or still alive.

2.4 Simulation of Networked Control Systems

GTSNetS was extended to implement the simulation of networked control systems. A networked control system consists of a set of control nodes having sensing, control and actuation capabilities and interacting using an overlapping network [16]. In such a system, any of the three main control loop tasks of sensing, control and actuation can be performed in a distributed manner. The use of wireless sensor networks for distributed control offers several benefits. It allows cost reduction and eliminates the need for wiring. Wiring could become costly and difficult in the case of a large number of control nodes needed for the sensing and control of a large process.

Control is an important application area of networked embedded systems and sensor networks in particular. In fact, distributed control has been a standard for large-scale processes, such as industrial automation and mobile robotics. From a sensor network point of view, a sensor network can be used in several ways in a control system.

2.4.1 Distributed Sensing with Centralized Control and Actuation

This approach consists of using an entire sensor network as the sensing entity in a control system. This allows monitoring a large plant that cannot be covered using a single sensor. It also allows for fault tolerance since sensor nodes can be deployed to have several nodes covering each part of the plant. Information collected by the different sensor nodes is fused and given as an input to the controller, which runs on a supervisor node outside of the sensor network. The controller generates a signal (action) that is applied to the plant by the actuator without involving the network. Figure 4 illustrates this approach. The arrows in the figure indicate the information flow.

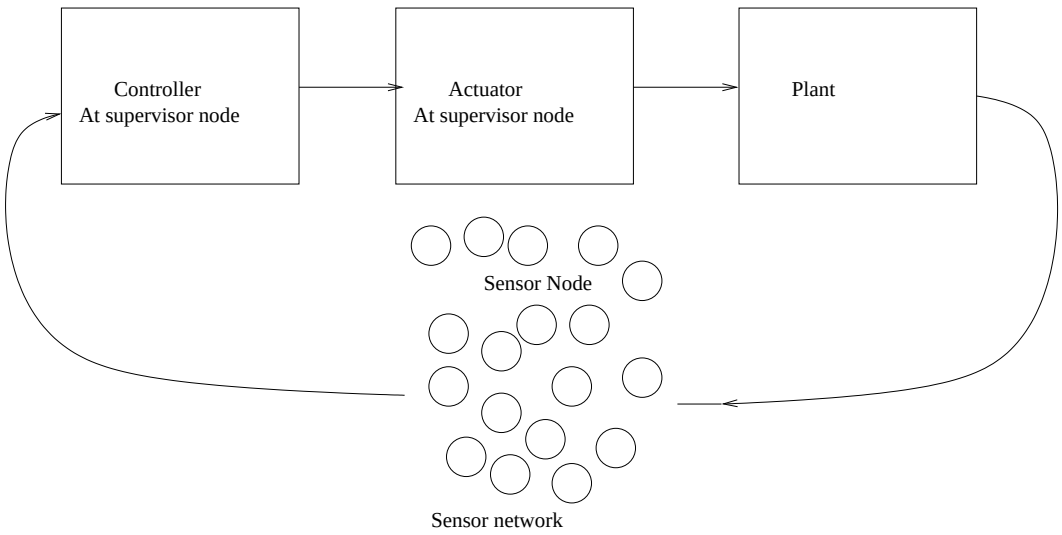

Figure 4: Distributed sensing with centralized control and actuation

2.4.2 Distributed Sensing and Actuation with Centralized Control

This approach is similar to the previous one. The main difference is that any corrective action on the plant is now applied by individual sensor nodes after receiving control commands from the controller. The controller is still run centrally at the supervisor node. Actuation messages are addressed either to all nodes in the network or to specific nodes as a function of their sensor readings. This could be the case, for example, when nodes in a specific region are reporting exceptionally high values. Figure 5 illustrates this approach.

2.4.3 Distributed Sensing, Control and Actuation

In this approach, the sensing, the control and the actuation are all performed inside the network. In this case, each control system acts as a sensor node. Each node can collect information about the plant, run control algorithms (act as a controller) and apply any necessary actions (actuator role). These nodes collaborate to control an entire system. However, each node monitors and actuates a specific plant that is a part of a larger system. In this case, each sensor node contains a controller, an

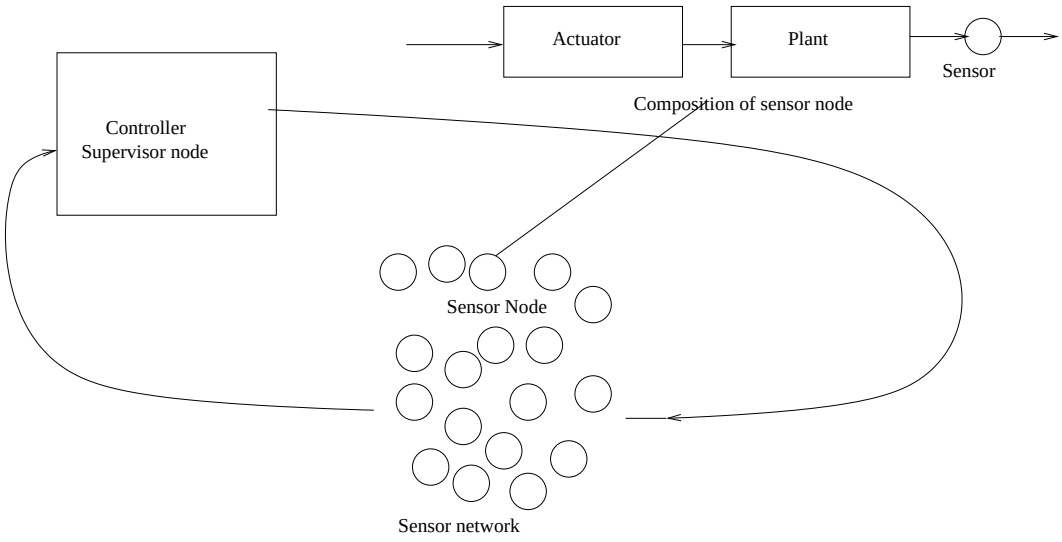

Figure 5: Distributed sensing and actuation with centralized control

actuator and a plant in addition to its normal components. In this approach, the sink node (base station) does not participate in the control process. It plays its traditional roles of collecting information, storing it and relaying it to the outside world when necessary. Figure 6 illustrates this approach.

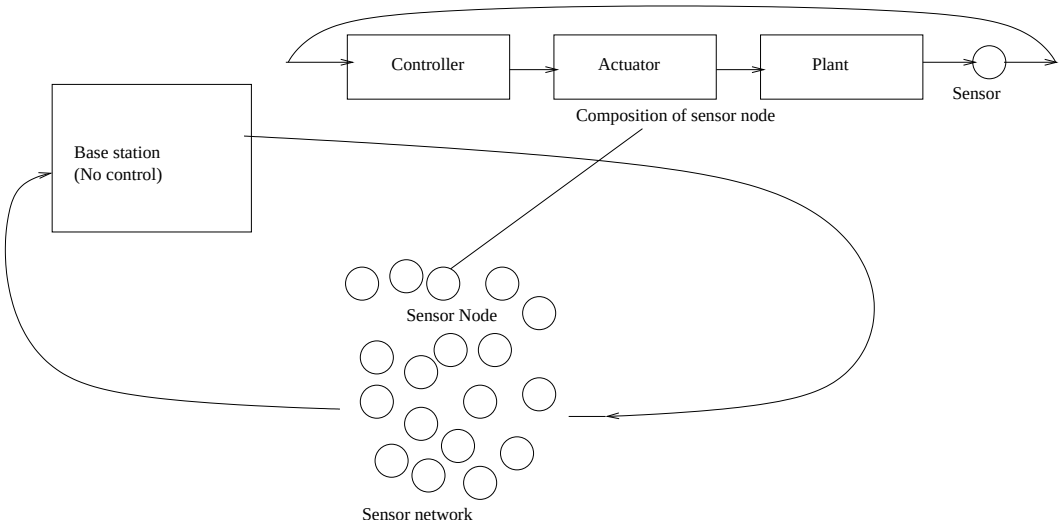

Figure 6: Distributed sensing, control and actuation

2.4.4 Hierarchical Distributed Sensing, Control and Actuation

This approach is similar to the previous one, except that a supervisor node can now affect the parameters of the control algorithm executed on individual sensor nodes. It can, for example, change the reference value at individual nodes or load a new control algorithm depending on the current state of the plant as reported by the individual nodes. It can also change the parameters to adjust to changing network conditions. For example, if the energy level becomes low at some of the nodes, these nodes can be asked to sample at a lower rate. Figure 7 illustrates this approach.

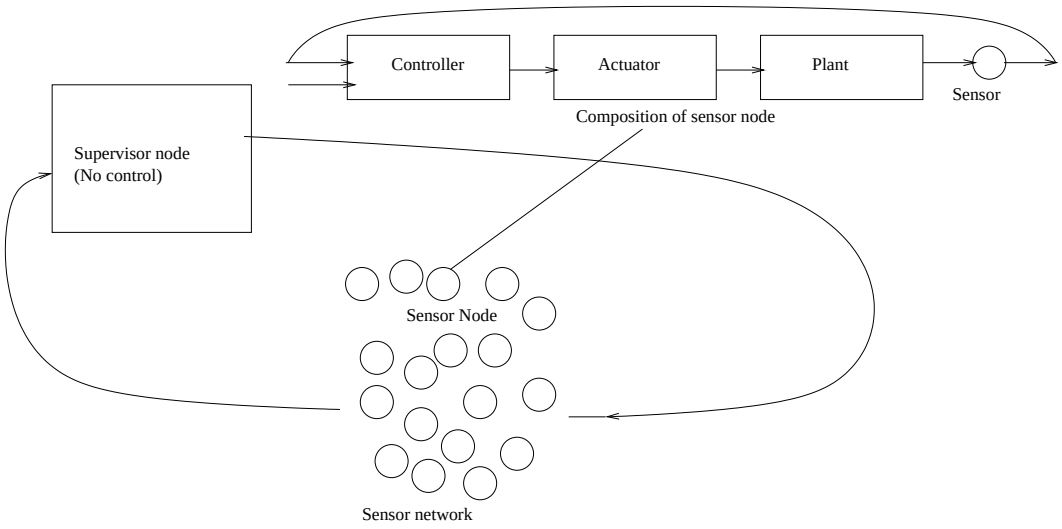

Figure 7: Hierarchical distributed sensing, control and actuation

To implement these different control architectures, several classes were added to *GTSNetS*. Due the modularity of *GTSNetS*, the addition of these classes does not have any impact on the existing classes. It is facilitated by the possibility of code reuse in *GTSNetS*. The first class (`class Plant`) models a plant. This class is derived from the real sensed object class that is already in *GTSNetS*. The main difference is that the plant can receive a command (signal) from an actuator. A second class implements the actuator (`class Actuator`). An actuator has a plant object associated with it. It can act on the plant in several manners depending on the commands received

from the controller. Several methods of actuations are implemented. The user can choose from these methods or implement his own if necessary. The controller is modeled using an application class (`class ApplicationSNController`) derived from the sensor network application class. Each controller has a specific actuator object attached to it. This actuator object performs actions on the plant depending on the control commands received from the controller application.

The different control architectures are implemented by attaching some or all of these classes to individual nodes. To implement the first architecture (distributed sensing with centralized control and actuation), we attach a plant object to each sensor node. However, since these nodes do not perform the controller and actuator roles, they each have a regular sensor network application. No actuator object is attached to the sensor nodes. A controller application is attached to the supervisor node. An actuator object is attached to this controller. This actuator acts on the plant object, which is sensed by the sensor nodes.

In a similar way, the second architecture is implemented by attaching a controller application to the supervisor node. However, this controller application gives commands to actuators, which are attached to individual sensor nodes. Each actuator acts on the plant object attached to its sensor node.

In the third architecture, a controller application is attached to each sensor node. This application commands an actuator that acts on a plant, which is attached to the same node. The sink node does not have any role in a fully distributed control application such as this case.

The fourth architecture is implemented in a similar way to the third one, except that a controller application is now attached to the supervisor node. This application can modify the control algorithms or other parameters at the individual sensor nodes depending on the information they supply.

2.5 Experiments

The experiments reported here are described to show how *GTSNetS* may be used to study the performance of a given sensor network under a specific configuration, and therefore, to allow for comparisons between different architectural choices. This is done through an example simulation. This section also reports results from a set of experiments designed to demonstrate the scalability of *GTSNetS* and its capability of simulating very large sensor networks.

2.5.1 An Example Simulation: Distributed Bayesian Fault-Tolerance Algorithm

In this subsection, *GTSNetS* is used to simulate an existing fault-tolerance algorithm [49] and verify that the obtained results correspond to the results published by the original authors. The original results reported in [49] were obtained using a MATLAB simulation and were found to agree in most part with the theoretical values. The comparison between our results and the original ones helps confirm the accuracy of *GTSNetS* models. The simulated algorithm provides fault-tolerance for event-region detection using wireless sensor networks. Nodes are tasked to detect when a specific event is present within their sensing range. Such an event could be detected through the presence of a high concentration of a chemical substance. Each node first determines if its sensor reading indicates the presence of an event before sending this information to its neighbors or to a sink node. However, in case of failure the sensor can produce a false positive or a false negative. That is, a high reading indicating an event occurred when it did not or a low reading indicating the absence of event when one occurred.

The solution presented in [49] considers a sensor reading of a high value as an indication of the presence of an event, while a low value is considered normal. It relies on the correlation between the node reading and the readings of its neighbors to detect faults and take them into account in the final decision process. The following

binary variables are used to indicate if a node is in an event region (value 1) or in a normal region value (value 0):

- T_i: indicates the actual state at the node (in an event region or not).
- S_i: indicates the state as obtained from the sensor reading. It could be wrong in the case of failure.
- R_i: gives a Bayesian estimate of the real value of T_i using the S_i values of the node and its neighbors.

It is assumed that all the nodes have the same uncorrelated and symmetric probability of failure, p:

$$P(S_i = 0|T_i = 1) = P(S_i = 1|T_i = 0) = p \tag{14}$$

The binary values S_i are obtained by placing a threshold on the reading of the sensor. The sensor reading when in an event region and when in a normal region are considered to have means of m_f and m_n, successively. The error term is modeled as a Gaussian distribution with mean 0 and standard deviation σ. In such a case, p is computed as follows using the tail probability of a Gaussian distribution:

$$p = Q(\frac{m_f - m_n}{2\sigma}) \tag{15}$$

Assume each node has N neighbors. Define the evidence $E(a, k)$ as the event that k of the N neighboring nodes report the same conclusion $S_i = a$. Using the spatial correlation, it can be proven that:

$$P(R_i = a|E_i(a, k)) = \frac{k}{N} \tag{16}$$

Each node can now estimate the value of R_i given the value of $S_i = a$ and $E_i(a, k)$. This is given by:

$$P_{aak} = P(R_i = a|S_i = a, E_i(a, k)) = \frac{(1 - p)k}{(1 - p)k + p(N - k)} \tag{17}$$

$$P(R_i \neq a | S_i = a, E_i(a,k)) = 1 - P_{aak} = \frac{p(N-k)}{(1-p)k + p(N-k)} \tag{18}$$

Three decision schemes can be used:

- Randomized: determine the values of S_i, k and P_{aak}; generate a random number $u \epsilon (0,1)$; if $u \leq P_{aak}$, then set $R_i = S_i$, else set $R_i \neq S_i$.
- Threshold: a threshold θ is fixed in advance; determine the values of S_i, k and P_{aak}; if $\theta \leq P_{aak}$, then set $R_i = S_i$, else set $R_i \neq S_i$.
- Optimal threshold: determine the values of S_i, k; if $k \geq \frac{N}{2}$, then set $R_i = S_i$, else set $R_i \neq S_i$.

It has been proved in [49] that the optimal value of θ in the threshold scheme is $1 - p$, which is equivalent to using the optimal threshold scheme. We therefore study only the randomized and the optimal threshold schemes. Several metrics have been developed to evaluate the performance of this Bayesian solution under different settings. These metrics include:

- Number of errors corrected: number of original sensor errors detected and corrected by the algorithm.
- Number of errors uncorrected: number of original sensor errors undetected and uncorrected by the algorithm.
- Reduction in errors: overall reduction in number of errors, taking into account the original errors and the ones introduced by the algorithm.
- Number of errors introduced by the solution: number of new errors introduced by the algorithm.

A full description of these metrics as well as their theoretical values can be found in [49].

To simulate this particular solution, only one class (`class ApplicationSNFToler`) was added to *GTSNetS*. The modularity of *GTSNetS* allows to reuse without any change all the other modules: computing unit, communication unit, sensing unit containing a chemical sensor with the appropriate accuracy model and sensed object.

A sensor network of $1,024$ nodes deployed in a uniform grid in a region of 680 meters by 680 meters is simulated. Neighboring nodes are separated by 20 meters. The communication range is set to 23 meters. Each node takes into account its immediate neighbors (nodes within its communication range) in its decision scheme. One source (sensed object) was placed at the lower left corner of the region of interest. The sensing range was set to 93 meters. These numeric values are only different from the ones in [49] by a scaling factor.

Figures 8 and 9 give some of the performance metrics results for the optimal threshold and randomized schemes for various fault rates. This results were obtained by averaging over $1,000$ runs. It can be seen that both decision schemes correct a high percentage of original errors (about 90% for the optimal threshold and 75% for the randomized scheme at 10% failure rate). These graphs are in agreement with the ones reported in the original paper [49] and with the theoretical values, which increases the confidence in the correctness of *GTSNetS*. The simulations in [49] were conducted using MATLAB and did not take into account many of the real-life communication aspects and energy constraints that are modeled in *GTSNetS*.

Clearly, the optimal threshold scheme performs better than the randomized scheme. This is expected and is in accordance with the findings in [49]. However, the randomized scheme has the advantage of giving a level of confidence in its decision to set $R_i = S_i$ or not. This confidence level is given by P_{aak}. This is not possible in the case of the optimal threshold scheme.

It is worth noting, here, that all the different simulations here took a relatively short time, which demonstrates the efficiency of *GTSNetS* in terms of execution time.

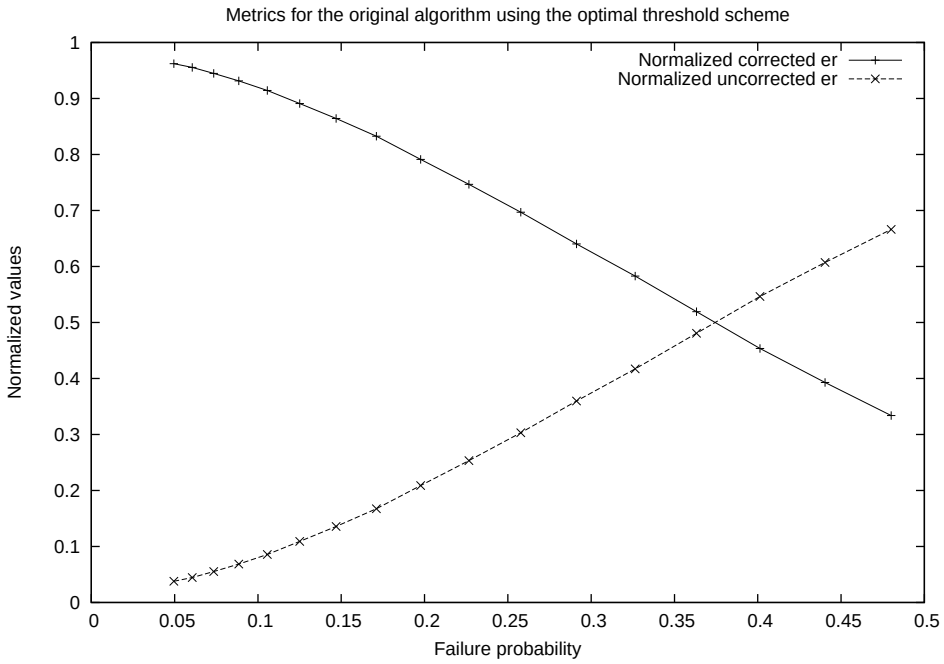

Figure 8: Performance metrics for the optimal threshold scheme

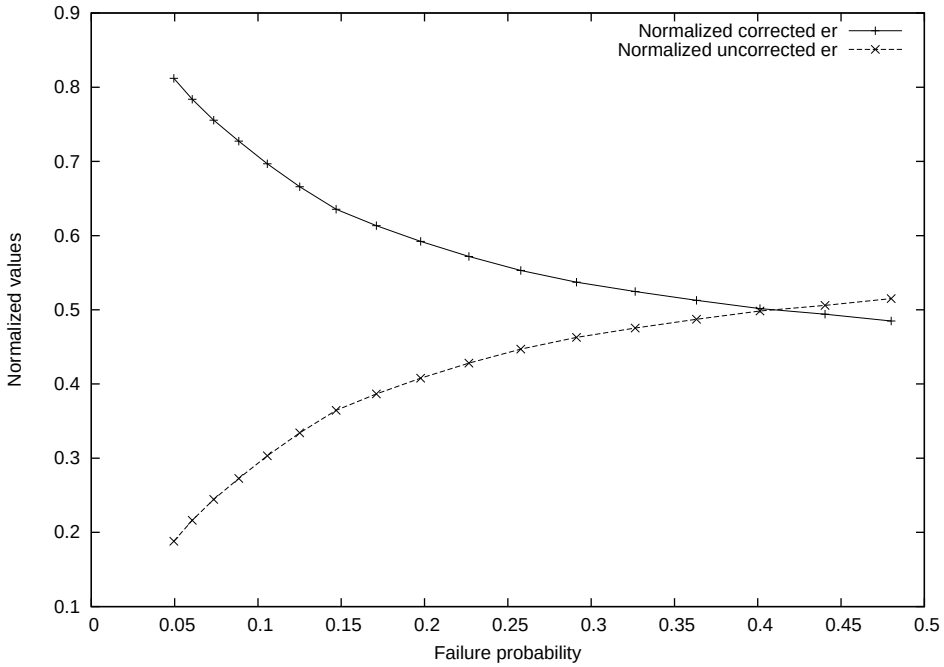

Figure 9: Performance metrics for the randomized scheme

In fact, for every set of simulation parameters, $1,000$ runs were completed in about 5 minutes. The simulator scalability is further investigated in the next subsection.

2.5.2 Scalability Testing

A set of tests was conducted to determine the capability of the simulator to simulate large sensor networks. These tests use the scenario below but with varying number of nodes and size of the deployment region.

1. Sensor Nodes are uniformly distributed in a square region: each sensor node has 4 neighbors in its communication range. Every two neighboring nodes are separated by a distance of 20 meters.

2. The sink node is located at the lower left corner of the deployment region.

3. Each sensor node has a battery with an initial energy of 2 Joules.

4. Each senor node has a temperature sensor.

5. Each node has a communication range of 21 meters.

6. Typical energy consumption parameters given in [11] were used for the energy models.

7. Directed diffusion is used as a routing protocol.

8. The sink node floods the network with an interest (request) at the beginning of the simulation. This interest has the following characteristics: interest in temperature from nodes in the 400 meters per 400 meters upper left subregion of the deployment area every 30 seconds throughout the lifetime of the nodes.

9. The sink node declares the network dead if it does not receive any message for a period of 500 seconds.

The tests start by simulating networks of thousands of nodes deployed in a small region and gradually increase the number of nodes while increasing the size of the region of deployment until reaching the limit that can be handled by the simulator. The simulator reaches the limit if the simulation does not finish properly due to insufficient virtual memory on the simulation platform.

These tests showed that in this scenario, *GTSNetS* can simulate networks of up to several hundred thousands of nodes. In fact, the largest simulated network in our experiments consists of a network of $200,000$ sensor nodes.

Figure 10 plots the network lifetime versus the number of sensor nodes. It shows that globally the lifetime decreases when the network get larger. This result is expected, since larger networks result in more hops on average per message transmitted. This means that more and more nodes lose energy while relaying messages containing data collected by other nodes, since sensor nodes collecting data are farther from the base station. In addition, the farther away are nodes collecting data, the greater is the number of possible paths that deliver the sensed data to the base station. There is no path reinforcement, which results in selection of some paths among all the available ones, so the data is sent on all the possible paths. The multiplication of paths induces higher energy consumption per message at the nodes closer to the base station. This in turn lowers the lifetime of these nodes and of the overall network.

Figure 11 plots the amount of memory used by the simulator as a function of the size of the simulated network. It can be seen clearly that *GTSNetS* is capable of simulating large networks of several hundred thousand nodes on machines having less than 2 GB of memory. Even though the used memory increases with the network size, it remains relatively small, and the memory requirement can be met by a typical desktop computer or workstation.

Figure 12 gives the execution time as a function of the network size. The execution time here is defined is the real clock time necessary to finish the simulation. As

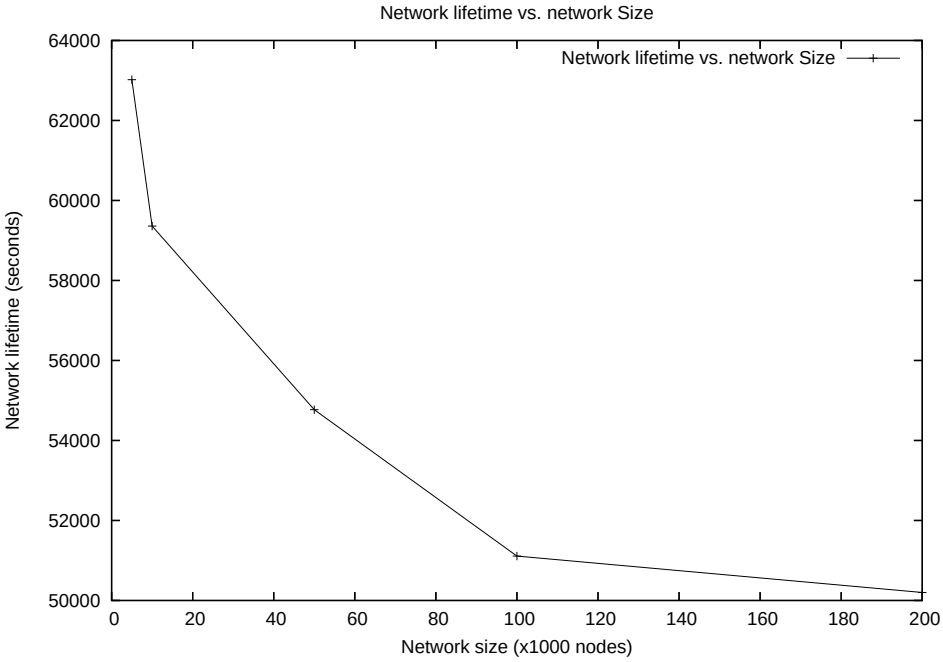

Figure 10: Network lifetime vs. network Size

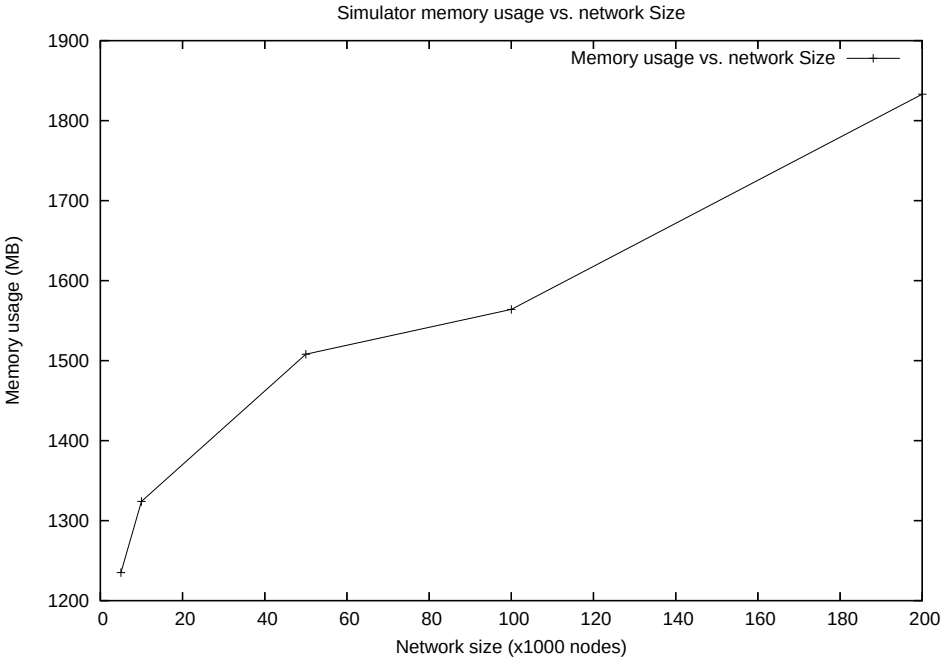

Figure 11: Simulator memory usage vs. network size

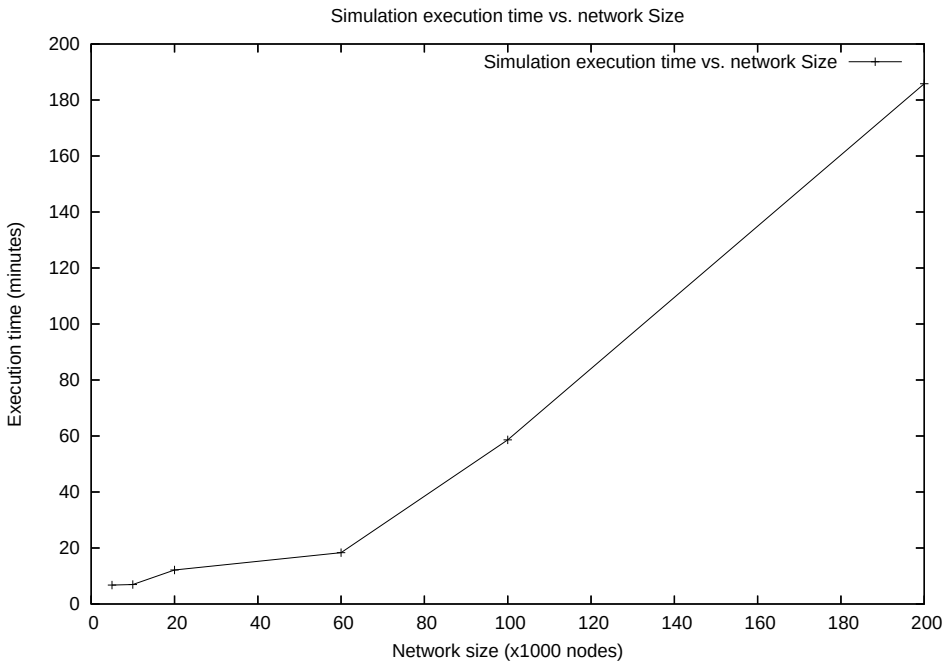

Figure 12: Simulation execution time vs. network size

expected, this time increases as the network size increases. However, the execution time remains very low even for these large networks: the simulator requires less than 7 minutes to simulate a network of $10,000$ nodes and only about 3 hours to simulate a network of $200,000$ sensor nodes.

2.6 Conclusion

This chapter described the design and implementation of *GTSNetS* and illustrated how it can be used to study sensor networks. One of the main benefits of *GTSNetS* is its scalability to network size. In fact, it has been demonstrated, through simulation results, that it can be used to simulate networks of several hundred thousand nodes. Other important features of *GTSNetS* include the fact that it is modular and easily extensible by users to implement additional architectural designs and models. It also supports the simulation of networked control systems having sensing, control and actuation capabilities which have been lacking in other sensor network simulators.

GTSNetS is extensively used in the coming chapters to evaluate the algorithms proposed to handle the global ID assignment and distributed detection problems in sensor networks.

GTSNetS is distributed under the GNU General Public License and can be obtained from the web page at `http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/download.htm`.

CHAPTER III

DISTRIBUTED GLOBAL ID ASSIGNMENT FOR SENSOR NETWORKS

The first algorithm contributed in this work handles the global ID assignment problem for wireless sensor networks. The need for global ID assignment is motivated in Section 3.1. The related work is discussed in Section 3.2. In Section 3.3, the proposed algorithm is discussed for the case of sensor networks with synchronized deployment. The theoretical analysis of the proposed algorithm is presented in Section 3.4. Section 3.5 discusses extensive simulation evaluation of the algorithm. The algorithm is extended in Section 3.6 to handle the case of sensor networks with asynchronous deployment. Section 3.7 presents simulation results for the extended algorithm. Section 3.8 discusses the cost and benefit of using our ID assignment algorithm. The chapter is concluded in Section 3.9.

3.1 Motivation

The communication range of individual sensor nodes is generally limited, and communication is often carried out in a multi-hop manner. There is a need to have a unique identifier in the header of every unicast packet. In fact, routing protocols need to uniquely identify the final destination as any node in the network can be a potential destination. Several routing protocols use attribute-based routing and therefore can use attributes as global identifiers. However, even these protocols require the existence of unique IDs at a local level. This is the case for directed diffusion [44] and geographical routing protocols such as the one proposed in [112]. Network-wide

unique IDs are beneficial for administrative tasks requiring reliability, such as configuration and monitoring of individual nodes, and download of binary code or data aggregation descriptions to sensor nodes [30]. Network-wide unique IDs are also required when security is needed in sensor networks [46]. Several MAC protocols requiring the preexistence of network-wide unique IDs have also been proposed for sensor networks [113].

Assumption of the preexistence of network-wide IDs is not realistic in the case of sensor networks. The preexistence of network-wide global IDs requires hard-coding these IDs on nodes prior to the deployment. This is costly in terms of time and effort when a network contains thousands to hundreds of thousands of nodes. Another alternative is to have MAC addresses that are unique for every manufactured sensor node, as is the case for Ethernet cards [103]. This is not a desirable approach because of the coordination it requires and the fact these IDs would have to be lengthy and therefore costly to use in packet headers.

An obvious ID assignment strategy is to have each node randomly choose an ID such that the probability of any two nodes choosing the same ID is very low. However, for this probability to be low, we need the IDs to be very long, which is again costly in terms of energy [97]. An ID assignment solution should, ideally, produce the shortest possible addresses because sensor networks are energy-constrained. The usage of the minimum number of bytes required is motivated by the need to limit the size of transmitted packets, in particular the header. In fact, communication is usually the main source of energy drain in a sensor node [91]. For this reason, sensor networks are designed to limit the amount of data transmitted, for example through data aggregation. This reduces the payload of transmitted packets, which makes the header size even more significant.

In this chapter, we introduce an algorithm that assigns unique IDs to sensor nodes using only the minimum number of bytes required to assign each node in the network

a unique ID. The algorithm does not assume the pre-existence of any type of identification and scales well with the size of the network. We also do not assume the existence of any specific communication protocol. In particular, the preexistence of a specific collision avoidance mechanism is not assumed. The algorithm handles collisions through avoidance and recovery. Collisions are avoided through the scheduling of transmissions at random times. If collisions occur, they are detected through a confirmation mechanism, and recovery is performed by retransmitting colliding packets. Our algorithm can handle the case of asynchronous wake-up. Nodes can join the network after the execution of the algorithm and still obtain a unique ID in an energy efficient manner. The handling of asynchronous wake-up is particularly important in the case of sensor networks where many nodes may be in a sleep state during the initialization phase for the purpose of energy saving and network lifetime optimization [50].

The algorithm can be divided into three main phases. In the first phase, a tree structure is established and, at the same time temporary long IDs are assigned. These temporary IDs are used for reliable communication during the remaining two phases. In the second phase, the size of sub-trees is reported bottom-up from leaf nodes to the root. In the third phase, the final short IDs are assigned. Each participating in the initial phase of the algorithm request additional spare IDs that are used to locally assign unique IDs to neighboring nodes that asynchronously join the network.

We analytically prove the correctness and termination of the algorithm. We also assess its performance in terms of the execution time and the probability that a node is left without an assigned ID at the end of the algorithm.

3.2 Related Work

In general, network-wide unique addresses are not needed to identify the destination node of a specific packet in sensor networks. In fact, attribute-based addressing fits

better with the specificities of sensor networks [32]. In this case, an attribute such as node location and sensor type is used to identify the final destination. However, different nodes can have the same attribute value, in particular in the same neighborhood. Thus, there is a need to uniquely identify the next hop node during packet routing [6]. Furthermore, it is possible that two neighboring nodes have the same attributes. For instance, it is likely that some nodes will have the same location in a dense sensor network. In addition, the number of bits required to represent attribute information (for example the node geographical coordinates) may be large rendering this approach less attractive from a communication energy point of view [117].

In [72], the authors propose an algorithm that assign globally unique IDs. Like our algorithm, it uses a tree structure to guarantee the uniqueness of each ID. The algorithm is similar to the first phase of our algorithm. It starts with the sink node broadcasting a message that contains its ID and a parameter b given the size in bits of one-hop ID. Successive nodes choose a parent node among their neighbors that already have an ID. The node then randomly chooses an ID of size b bits and relays on its parent to guarantee no other node has chosen the same ID. The node then appends its chosen to ID to the ID of its parent to create a unique ID. The main difference between this algorithm and ours is that it does not use the network size to minimize the size of node IDs. Our algorithm, in contrast, not only assigns unique IDs but also guarantees that these IDs are of minimum length. This is is a considerable advantage considering that sensor networks are energy-sensitive.

Several schemes have been proposed to assign locally unique addresses in sensor networks. In [91], Schurgers, et al., developed a distributed allocation scheme where local addresses are spatially reused to reduce the required number of bits. The pre-existing MAC addresses are converted into locally unique addresses. Each locally unique address is combined with an attribute-based address to uniquely determine the final destination of a packet. This use of locally unique addresses instead of global

addresses does not affect the operations of the existing routing protocols. This solution assumes the preexistence of globally unique addresses, which our algorithm does not assume. Our solution can be used to assign these global addresses prior to the use of the method in [91]. Our proposed solution allows for asynchronous wake-up. When a new node joins the network, it chooses a random address and shares it with its neighbors. The neighbors compare the new address with the addresses of other neighbors to detect and resolve any conflicts.

In [6], Ali, et al., proposed an addressing scheme for cluster-based sensor networks [38]. To prevent collisions, nodes within the same cluster are assigned different local addresses. Non-member one-hop and two-hop neighbors must also have different local addresses to avoid the hidden-terminal problem. The network is divided into hierarchical layers where the number of layers increases with the number of nodes in the network. Global IDs are obtained by concatenating the local address and the addresses of the head nodes of the different layers. This solution suffers from the fact that the address size increases with the number of layers as 6 bits are added for each layer. This makes this solution less attractive due to the energy cost of using global IDs in the case of large sensor networks. In addition, this solution can be used only with cluster-based routing and does not extend to the case of multi-hop routing [39].

In [30], Dunkels, et al., developed a spatial IP addressing scheme using node location. The (x, y) coordinates of a node are used as the two least significant bytes of its spatial IP. This solution is particularly attractive since it can facilitate the interaction between sensor networks and other types of networks. However, it also suffers from the large size of generated addresses leading to higher overhead. It also requires the existence of a localization mechanism since it assumes that nodes are location-aware.

In [117], the authors propose a local ID assignment scheme where address conflicts are resolved in a reactive way. Nodes randomly choose an address that is likely to be

unique within a 2-hop neighborhood. No conflict resolution is performed until nodes need to enter in a communication. For instance, interest broadcasting in directed diffusion can be used to resolve conflicts. In this case, the sink node discovers the existence of identical IDs between its immediate neighbors. The conflicting nodes are notified and choose new IDs. Each node that forwards the interest message uses the response messages to detect ID conflicts among its neighbors. The delaying can help save energy by avoiding any unnecessary conflict resolution. In particular, if two neighbors have the choose the same local ID but are never active at the same time, resolving such a conflict amounts to a waste of energy resources. However, resolving ID conflicts reactively can be problematic if the sensor network requires time-sensitive exchange of information, since messages can be delayed to resolve an ID conflict.

In [66], Motegi et al. propose an on-demand address scheme. To reduce the number of bits required to represent addresses and the number of control messages needed to establish these addresses, the authors propose to assign temporary addresses to nodes detecting an event on an on-demand basis. When a node becomes active (detects an event), it chooses a random network-wide ID and sends a route request to the sink node using the AODV protocol [77]. The intermediate and the sink node perform a conflict resolution. Once the node communication with the sink node terminates (the node is no longer active), its address goes back to the free address pool and can be used by newly active nodes. This approach can effective reduce the number of bits required for globally unique IDs. However, it assumes the pre-existence of some of form of network-wide unique IDs (e.g., MAC or IP addresses). The pre-existing addresses are needed to establish locally unique and permanent addresses used by the proposed algorithm for neighbor identification and collision avoidance.

In [55], the authors present the GREENWIS algorithm designed to assign group IDs. The objective is to assign group IDs rather than individual node IDs. Neighboring nodes share the same group ID, which allows the number of bits required to

represent each to be much smaller. The main problem with this approach is that it can be used to provide functionalities such as collision avoidance as all neighbors are assigned the same ID.

The ID assignment problem is related to the overall sensor network initialization. Initialization can be viewed as the mechanism needed for individual sensor nodes to become an integral part of a sensor network. When a sensor network is initially deployed, there is no established structure to allow nodes to efficiently communicate [50]. The initialization has the objective of transitioning from this unstructured state to a structured network and the establishment of a MAC protocol to allow efficient information dissemination. An important way to structure sensor nodes into a network is through the use of clustering techniques [47, 99, 50]. Many of the clustering algorithms that have been proposed for sensor networks assume the pre-existence of unique node IDs. For example, in the approach proposed in [50] a newly active node that wants to join the network waits for messages from neighboring dominators (cluster heads) before trying to become a dominator. A unique ID is needed to distinguish between the different dominators in the neighborhood. The ID assignment algorithm proposed here can be used as part of the overall network initialization phase. In this way, the initialization does not need to require the pre-existence of node IDs. Like the work in [50], our approach does not assume that all nodes wake-up at the same time. We also do not assume the existence of a specific collision avoidance protocol.

3.3 Unique ID Assignment Algorithm

We present a distributed algorithm that assigns globally unique IDs to sensor nodes. Initially, we assume that all nodes are awake during the execution of the algorithm. This assumption is relaxed later in this chapter to accommodate a dynamic network where nodes can join the network at any time during the execution of the algorithm or after its termination. The algorithm can be divided into three main phases. In

the first phase, the objective is to assign temporary unique identifiers in the form of potentially long vectors of bytes. A tree structure rooted at the node initiating the algorithm is established during this phase.

In the second phase, the temporary identifiers are used to reliably compute the size of each sub-tree and report it to the parent node. This process is done for each sub-tree from leaf nodes until the root node. At the end of this phase, the initiator knows the total size of the network. This allows the initiator to compute the minimum number of bytes required to give a unique ID to each node in the tree.

The third phase consists of assigning final IDs to each node in the network going from the root to the leaf nodes. These different phases are now described in detail.

3.3.1 Phase 1: Tree Building and Temporary ID Assignment

In this phase, temporary IDs are assigned and a tree structure is established. The temporary ID of a particular node is a vector of bytes that uniquely identifies it. The temporary ID of a child node has one byte more than that of its parent. We assume a network density, such that no node has more than 256 neighbors. However, for networks of higher density, temporary IDs can be modified to be vectors with elements of 2 or 3 bytes as needed. The algorithm starts with the initiator node, typically the base station, choosing its temporary ID to contain one byte of value 0, and broadcasting an initialization message of type 1. Each node receiving an initialization message for the first time considers its parent to be the sender of the message and initializes its temporary ID to that of its parent node.

The receiving node then randomly chooses a 4-byte integer and sends it in a message of type 2 to its parent node. This message also contains a retry counter. Upon reception of a new message of type 2, the parent node checks if any other child node had already chosen the same random number. If so, a reinitialization message of type 3 is sent to the child node. If no other child had chosen the same number, the

parent node sends a message containing an assigned ID of one byte that is different from the ones sent to other children nodes. This message is of type 4. The reception of this message is confirmed to the parent by a confirmation message of type 5.

After receiving the message of type 4 containing the 1-byte unique child ID, this byte is added at the end of the temporary ID. The child node then schedules the sending of an initialization message at random time uniformly distributed between $timeWait$ and $2 \times timeWait$. At the scheduled time, the node sends an initialization message of type 1 and waits for a certain amount of time ($5 \times timeWait$) to hear from any potential children. If any child responds within this period, the previous procedure of assigning one byte ID repeats itself. If no child responds, the node considers itself a leaf node.

All messages except the ones of type 1 are exchanged in a reliable way. A message of type 2, which contains the random 4-byte integer chosen by a child node, is confirmed by the reception of a message of type 3 (reinitialization message) or type 4 (containing a 1-byte assigned ID). A message of type 3 is resent if the parent node receives a second message of type 2 with the same 4-byte ID. A message of type 4 is confirmed through the reception of the confirmation message of type 5. If a message is not confirmed within a random period chosen uniformly between $timeWait$ and $2 \times timeWait$, the message is resent. The node keeps checking for a confirmation and resending until the message is confirmed.

Figure 13 illustrates the messages exchanged between a parent node and a child node during phase 1 when no reinitialization message is sent. A reinitialization message makes the child node resends the message of type 2. At the end of this exchange, the child node has a temporary ID that is 1 byte longer than the one of its parent node. The child node then sends its own message of type 1. Algorithm 3.3.1 gives the pseudo code of the first phase. Note that at the end of this phase every node knows the temporary ID of its parent node. The parent ID is equal to the node ID without

the last byte.

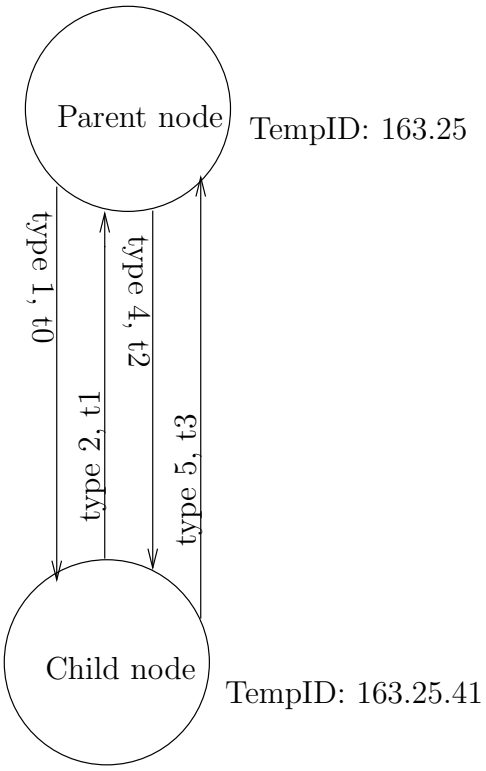

Figure 13: One step of phase 1 with no reinitialization message

3.3.2 Phase 2: Collecting the Sub-Tree Sizes

In this phase, nodes report their sub-tree sizes from the leaf nodes to the root node. The sub-tree size of a particular node is the number of nodes contained in the tree rooted at that node at the end of phase 1. A node that is declared leaf at the end of phase 1 considers its sub-tree size to be 1 and sends it as a message of type 6 to its parent. A non-leaf node waits until it receives sub-tree sizes from all of its children nodes before sending its sub-tree size to its parent. Sub-tree size messages are confirmed by the parent node with a confirmation message of type 7. Figure 14 illustrates the message exchange during phase 2 to collect the sub-tree sizes.

When the initiator receives sub-tree size messages from all of its children, it knows the total number of nodes in the network. This total is used to compute the minimum

Algorithm 1 Phase 1: Temporary ID Assignment

```
ChildB := 1
idConfirmed := false
if initiator is true then
    tempId := 0
    send an initialization message of type 1
end if
if receive message msg of type 2 then
    if a child already have same intId then
        send reinitialization message of type 3
    end if
    if no child already has same intId then
        add to children list
        choose a random time rt
        schedule checking for confirmation at rt
        send message of type 4 with ChildIdB
        ChildIdB := ChildIdB + 1
    end if
end if
if receive msg message of type 5 then
    if msg.dest = tempId then
        find ch, the corresponding child
        ch.tempIdConfirmed := true
        ch.sizeReceived := false
    end if
end if
if receive first message msg of type 1 then
    idAssigned := true
    tempId := msg.source
    choose a random 4-byte intId
    choose a random time rt
    schedule checking for confirmation at rt
    send message of type 2 with intId
end if
if receive msg message of type 3 then
    choose a different random 4-byte intId
    choose a random time rt
    schedule checking for confirmation at rt
    send message of type 2 with intId
end if
if receive msg message of type 4 then
    if idConfirmed is true then
        update timeWait
        resend message of type 5
    end if
    if idConfirmed is not true then
        update tempId and idConfirmed
        update timeWait
        choose a random time rt
        schedule sending message of type 1 at rt
        send message of type 5
    end if
end if
```

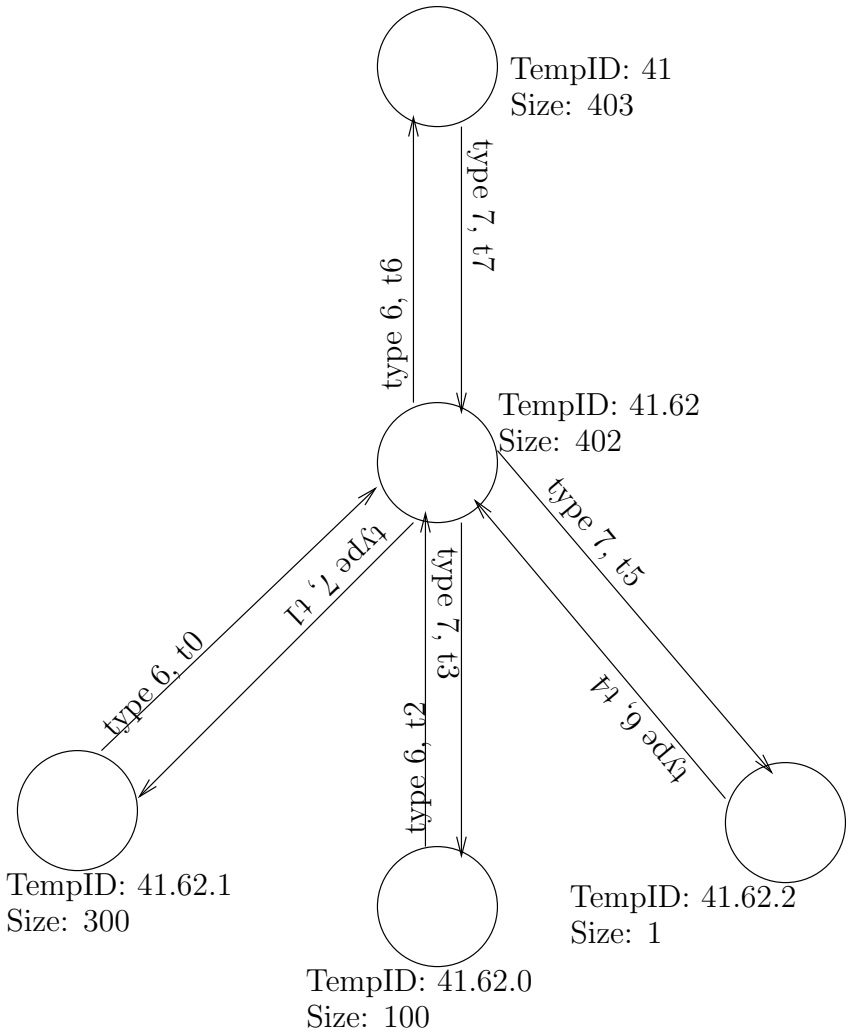

Figure 14: One step of phase 2

number of bytes needed to code a unique final ID for each node in the network. These IDs are assigned in phase 3 of the algorithm. Algorithm 3.3.2 gives the pseudo code of the second phase. Note that at the end of this phase every node knows its sub-tree size as well as the sub-tree size of each of its children nodes.

Algorithm 2 Phase 2: Sub-tree Sizes Collecting

```
subtreeSize := 1
sizeConfirmed := false
if receive msg message of type 6 then
    find ch, the corresponding child
    if ch.sizeReceived is true then
        resend message of type 7
    end if
    if ch.sizeReceived is not true then
        ch.subtreeSize and ch.sizeConfirmed
        subtreeSize := subtreeSize + ch.subtreeSize
        send message of type 7
        choose a random time rt
        schedule checking if all sub-tree sizes received at rt
    end if
end if
if leaf is true then
    choose a random time rt
    schedule checking for confirmation at rt
    send message of type 6
end if
if receive msg message of type 7 then
    if sizeConfirmed is not true then
        update sizeConfirmed
    end if
end if
if sub-tree size messages received from all children and initiator is not true then
    choose a random time rt
    schedule checking for confirmation at rt
    send message of type 6
end if
```

3.3.3 Phase 3: Final Unique ID Assignment

In this phase, the final unique IDs are assigned by each parent node to its children nodes starting from the root. Final IDs are coded using the same number of bytes (i.e., 1, 2, 3, or 4) for all nodes. The initiator is assigned an ID of 0. It sends a final ID message (message of type 8) to each of its children nodes. Each message contains a unique ID and the number of bytes to be used to code IDs. Final ID messages are confirmed with messages of type 9. Each node receiving a message of type 8 takes the ID it contains as its final ID and knows that a number of IDs starting from its ID and containing as many IDs as needed is reserved for the IDs of the nodes in its

sub-tree. Each non-leaf node receiving a final ID message confirms it and assigns IDs to its children nodes in a similar way.

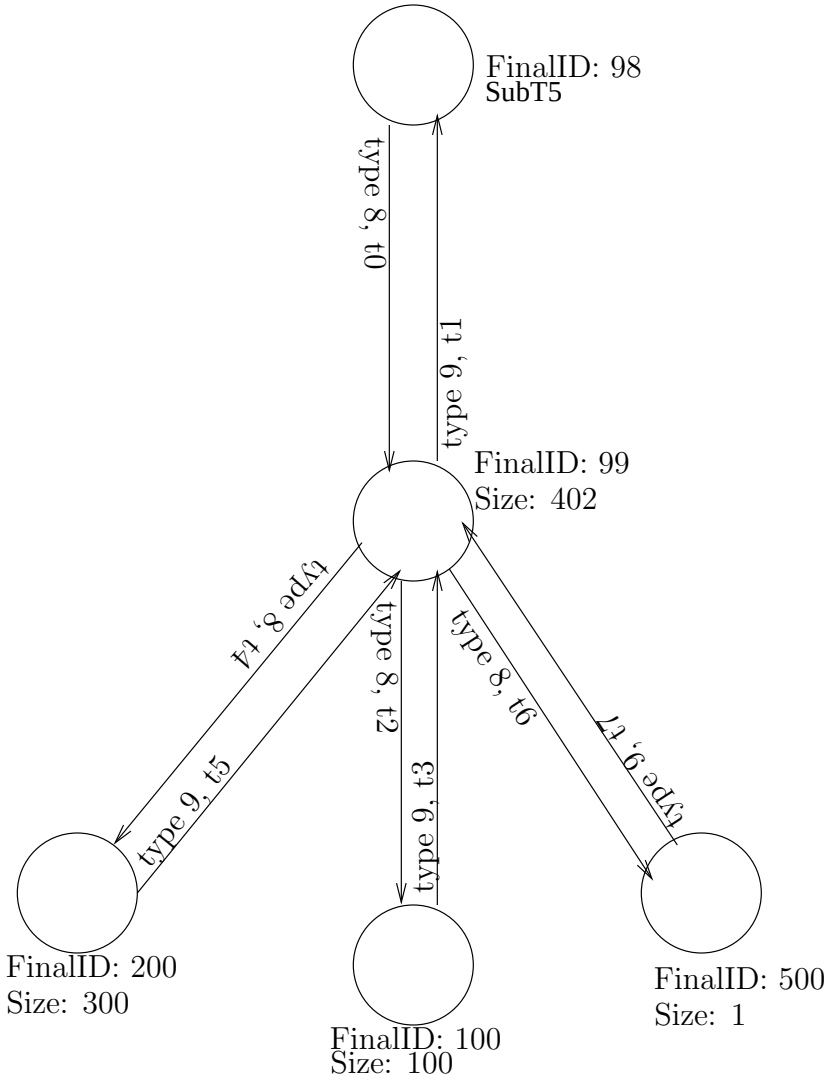

Figure 15: One step of phase 3

Figure 15 illustrates the message exchange during phase 3 to allocate the final IDs. Each node allocates its ID plus 1 to its first child and then allocates ID_i to the i^{th} child with $ID_{i+1} = ID_i + S_i$, where S_i is the sub-tree size of the i^{th} child. Algorithm 3.3.3 gives the pseudo code of the third phase. At the end of this phase, every node in the network knows its final ID. These final IDs are coded using the minimum number of bytes.

Algorithm 3 Phase 3: Final IDs Assignment

```
if sub-tree size messages received from all children and initiator is true then
    compute nbBytes, the number of bytes
    myId := 0
    idAssignedF := true
    currentId := 1
    choose random time rt
    schedule checking for confirmation at rt
    send message of type 8 to first child with currentId
end if
if receive msg message of type 9 then
    find ch, the corresponding child
    if ch.idFinalConfirmed is true then
        ignore
    end if
    if ch.idFinalConfirmed is not true then
        ch.idFinalConfirmed := true
        currentId := currentId + ch.subtreeSize
        if more nodes in the children list then
            choose random time rt
            schedule checking for confirmation at rt
            send message of type 8 to next child with currentId
        end if
    end if
end if
if receive msg message of type 8 then
    if idAssignedF is true then
        resend final ID confirmation message of type 9
    end if
    if idAssignedF is not true then
        idAssignedF := true
        myId := msg.minId
        currentId := myId + 1
        send message of type 9 to parent node
        choose random time rt
        schedule checking for confirmation at rt
        send message of type 8 to first child with currentId
    end if
end if
```

3.3.4 Collision Handling

Assuming a single channel, if a node n_s is transmitting a message to a node n_r, a collision occurs if n_r is already in the process of receiving from a different node. The algorithm does not assume the existence of any specific MAC address. In particular, no collision avoidance mechanism is required. Collision is handled in the sense that all messages except the initialization message (message of type 1) received by a node are confirmed by an acknowledgment message. Before sending a message, a node chooses randomly an integer number rn between 0 and $RANDMAX$, and waits for a time equal to $(1 + rn \div RANDMAX) \times timeWait$. If it does not receive the confirmation within the random waiting time, it resends the message and keeps doing so until receiving the confirmation.

The node adapts the parameter $timeWait$ to the traffic condition. In fact, this parameter is increased by half of its initial value ($timeWaitI$) every time an expected confirmation is not received, unless $timeWait$ has already reached an upper limit set to $5 \times timeWaitI$. Upon the reception of a message, $timeWait$ is reduced by half of $timeWaitI$, unless a lower bound, set to the initial value, is already reached.

For the message of type 1, it is assumed that every node has several neighbors. Each neighbor sends an initialization message at different times (randomly chosen after the first phase) to reduce the probability of collision. Therefore, a node has several possibilities of receiving an initialization message.

3.4 Theoretical Analysis

This section contains the theoretical evaluation of the unique ID assignment algorithm. In particular, the correctness of the algorithm is analyzed. We also prove that the algorithm terminates naturally and give an upper limit on the average energy consumption per node. Since the initial assignment messages (of type 1) are sent in a reliably, we also analyze the probability of a node being left out by the algorithm.

Such a node does not participate in the algorithm and is not assigned an ID.

In this section, we study the case of a synchronized wake-up where all nodes composing the network participate in the initial phase of the algorithm. In particular, the following properties are assumed. The case of asynchronous wake-up is handled in Section 3.6 later in the chapter.

Assumption 1 *All nodes are assumed to be awake during the initial phase of the algorithm.*

This assumption implies that no node is allowed to join the network after the initial phase of the algorithm. A node joining the network after all messages of type 1 have been sent by its neighbors cannot participate in the algorithm and will not receive an ID.

Assumption 2 *All nodes that participate in the initial phase of the algorithm are assumed to remain active until the termination of the algorithm.*

This assumption is needed to avoid that a parent or a child node waits indefinitely for a message for a node that is no longer active. As shown later in the theoretical and simulation analysis, this assumption is realistic since the algorithm terminates in a relatively short time. This implies that it is unlikely that a node will run out of energy during the execution of the algorithm.

The two assumptions above are relaxed in Section 3.6.

3.4.1 Model

The evolution of each node, except for the initiator, is modeled as a stochastic process with state space of $s = \{0, 1, 2, 3, 4, 5\}$. The different states are defined as follows:

1. State 0: A node is in state 0 if it did not yet receive any initialization message (message of type 1).

2. State 1: A node is in state 1 if it has already received an initialization message, is still waiting for its temporary ID to be confirmed by its parent node.

3. State 2: A node is in state 2 if its temporary ID has been confirmed by its parent node, but it did not yet send a message of type 1.

4. State 3: A node is in state 3 if its temporary ID has been confirmed by its parent node, it has sent a message of type 1, but did not yet send its sub-tree size message. This could be because it is still waiting to know if it is a leaf, or is still waiting for at least one child node to report the size of its sub-tree. It could also be during the period after receiving all sub-tree sizes, but the scheduled time to send its sub-tree message has not been reached

5. State 4: A node is in state 4 if it has already reported its sub-tree size to its parent node but is still waiting to receive its final ID.

6. State 5: A node is in state 5 if it has already received its final ID.

Clearly, state 5 is a stable state after which the node does not go back to any other state. It is also clear that a node can only go to the next higher state or remain in its current state. That is, for example, a node in state 3 can only go to state 4 or remain in state 3. The probability that a node changes its state depends on its current state as well as the states of the neighboring nodes. In fact, the neighbors influence the node state in several ways. A non-initiator node currently in state 0 can go to state 1 only if at least one of its neighbors is already in state 2. A non-leaf node in state 3 can change to state 4 only if all of its children nodes (a sub-set of its neighbors) are already in state 4. A non-initiator node currently in state 4 can go to state 5 only if its parent node is already in state 5. More generally the neighbors affect the probability of change in the sense that they can cause collisions if transmitting simultaneously. Collisions cause messages to be retransmitted and delay state changes.

We define the probability $p_i(t)$ as the probability that when the node is in state i at time t, it goes to state $i+1$ in the next step $(t+1)$ with i between 0 and 4. As stated earlier, the value of $p_i(t)$ depends on the current state of the node as well as the current states of its neighbors, in particular its parent and children nodes. Figure 16 gives the state diagram.

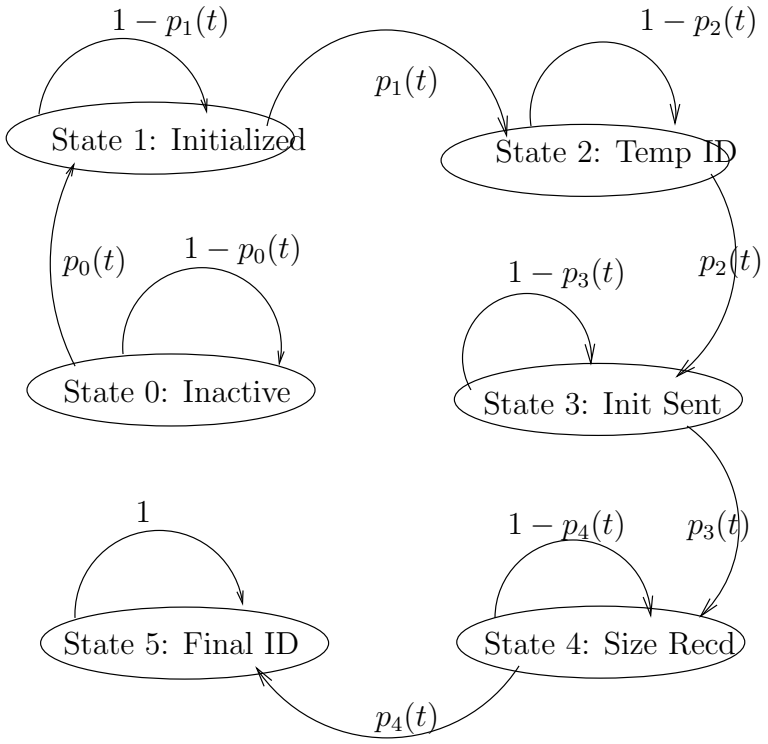

Figure 16: States diagram

3.4.2 Performance of the Algorithm

In this subsection, several properties of the algorithm are proved. In particular, the termination and correctness of the algorithm are studied. We also study the probability of a node not being assigned an ID at the end of the algorithm. This is a measure of the effectiveness of the algorithm. We also look at the energy cost of the algorithm.

The following lemma shows that if nodes do not die during the execution of the

algorithm, then the algorithm terminates.

Lemma 1 *Under Assumption 2, the algorithm terminates.*

Proof 1 *The statement in this lemma is equivalent to declaring that any node reaching state 1 in the states diagram will eventually reach state 5 if no node dies during the algorithm execution. This is clearly the case when the probabilities* $p_1(t)$*,* $p_2(t)$*,* $p_3(t)$*, and* $p_4(t)$ *are each greater than 0. Each of these probabilities is permanently equal to 0 only if a neighboring node with which the node interacts in the current state (parent or child node) is no longer alive. In fact, if such a node dies, the dependent node can continue sending a message indefinitely while waiting for a confirmation. It can also indefinitely wait for a size message (in case of parent node) or a final ID message (in a case of a child node). If no node in the network dies, each of the probabilities* $p_1(t)$*,* $p_2(t)$*,* $p_3(t)$*, and* $p_4(t)$ *does not remain equal to 0 all the time. Therefore, each node reaches the final state. The algorithm terminates when all leaf nodes reach the final state (state 5 in the states diagram).*

Before studying the correctness of the algorithm, we look at the possibility of two nodes with the same parent receiving the same temporary ID. This occurs only when two children of the same node choose the same 4-byte ID in first phase and respond simultaneously with messages of type 2 to the initialization message and their parent receives only one of the two messages. For the two nodes to end up with the same temporary ID, they need also to send simultaneously the confirmation message of type 5 and for their parent to receive one and only one of these messages.

We define P_{i2} as the probability of two nodes having two identical temporary IDs. As we can see, P_{i2} is very low because the occurrence of two nodes having two identical temporary IDs is conditioned on the joint occurrence of a successive number of independent events each having a very low probability. In fact, $P_{i2} \leq P_f$, where P_f is the probability of any two nodes in the network with the same parent choosing

the same 4-byte integer. It can be proved that for a network of n nodes each having no more than 256 neighbors, we have $P_{i2} \leq P_f \leq (n \times 256) \div 2^{32}$. For $n = 10,000$, we obtain $P_{i2} \leq 5.96 \times 10^{-4}$. The following lemma states the correctness of the algorithm.

Lemma 2 *Under Assumptions 1 and 2, each node receives a unique ID with a high probability of* $1 - P_{i2}$*, where* P_{i2} *is as defined above.*

Proof 2 *The proof comes from the nature of the assignment of temporary and final IDs. Since temporary IDs are assigned in a hierarchical way with children of same node all having different IDs, phase 1 ends with every node having a different temporary ID. The only exception is when two nodes receive the same temporary ID. In phase 3, each parent node, starting from the initiator, reserves a different set of final IDs for each of its children nodes having different temporary IDs to assign to its subtree. Therefore, every node having a unique temporary ID ends with a unique final ID. This proves the correctness of the algorithm.*

We now determine the probability that a message is successfully transmitted by the first trial and the probability of reception by the second trial. A message is not successfully transmitted if a collision occurs or a bit error prevents the successful interpretation of the message or the corresponding acknowledgment.

A collision is detected by the sender node, n_s, when it does not receive the corresponding confirmation message in a randomly predetermined time period. As explained in Subsection 3.3.4, the length of this time period is uniformly distributed between $timeWait$ and $2 \times timeWait$. If no confirmation is received, the message is resent at the end of this period. A collision occurs if the receiving node, n_r, is currently in the process of receiving a different message. It also occurs if a different neighbor of n_r broadcasts a message while the current message is being received. If the size of the current message is S bytes and the capacity of the radio is B kbps, the transmission (reception) time of the message is given by: $T_t = 8S \div (1,000 \times B)$.

Suppose that the transmission starts at t_0, the current message is not received (collision) if at least one of the other neighbors n_r transmits a message in the time interval $[t_0 - T_t, t_0 + T_t]$. If n_r has k neighbors, including the sender n_s, each neighbor transmits at most one message during each period of length tw_0, where tw_0 is the initial value of $timeWait$. Consequently, there are at most $k-1$ messages sent by the other neighbors. Each message is followed by a confirmation message except for a message of type 1 or when confirming a previous message from n_r. Therefore, there are at most $k-1$ confirmation messages and a total of $2 \times (k-1)$ messages. Assuming that all messages have approximately the same size S, the current message encounters a collision if its reception starts in one of at most $2 \times (k-1)$ transmission periods of length $2 \times T_t$.

A message cannot be correctly interpreted if it experienced a bit error, which has the same effect as a collision : the message must be retransmitted. If we consider a bit error rate of BER, it can be easily verified that the probability of a message and corresponding acknowledgment of size $2 \times S$ experiencing a bit error is given by: $P_e = 1 - (1 - BER)^{S \times 8}$.

The following lemma bounds the probabilities that a message is received successively after one or two trials.

Lemma 3 *If tw_0 is such that $P_c = 32 \times (k-1) \times S \div (1,000 \times B \times tw_0) \leq 1$, then:*

- *The probability of a message successfully received upon the first transmission is at least $P_1 \geq 1 - (P_c + P_e)$.*
- *The probability of a message successfully received within two transmissions is at least $P_2 \geq 1 - (P_c + P_e)^2$.*

Proof 3 *This follows from the fact that a collision occurs if the reception of the message starts during one of at most $2 \times (k-1)$ transmission periods each lasting $2 \times T_t$. Since each node sends at most one message in each time interval of length*

timeWait $\geq tw_0$*, we obtain the maximum collision probability:* $P_c = 4 \times (k-1) \times T_t \div tw_0 = 4 \times (k-1) \times 8 \times S \div (1,000 \times B \times tw_0) = 32 \times (k-1) \times S \div (1,000 \times B \times tw_0)$*. If we consider the events of bit error and collision as independent, the probability of a message not received in the first trial is* $P_c + P_e$*. Two unsuccessful transmissions must occur for the message not to be received by the second trial. Consequently, the probability of successful transmission by the first trial is given by:* $P_1 \geq 1 - (P_c + P_e)$*. In the same way,* $P_2 \geq 1 - (P_c + P_e)^2$*.*

This demonstrates that by appropriately setting tw_0 to limit the collision probability, we can guarantee a high probability of transmission of messages by the second trial. For a numerical example, we assume that the radio transmission rate is $B = 100kbps$, which is reasonable for current technology since transmission rate for MICAz motes for example is 250 kbps. We also assume that $k = 21$, the network having a density of 21. The message size S is function of the number of hops from the base station since each address is composed of one byte per hop. Let assume that at most $S = 100$. This limit holds even for large networks with low densities. Assuming that $BER = 2 \times 10^{-4}$ and $tw_0 = 2seconds$, then we have: $P_1 \geq 85.19\%$ and $P_2 \geq 97.81\%$. These values are conservative lower limits since most messages closer to the root node will be of much smaller size. As can be seen in the simulation results, realistic scenarios give much smaller values. For instance, P_1 is above 92% in all our simulation scenarios.

By following the same reasoning as above, we can bound the probability of a node not assigned an ID at the end of the algorithm. This occurs if the messages of type 1 (initialization messages) from all of its neighbors are lost. Since each of these messages is sent at a random time, we obtain the following lemma.

Lemma 4 *If k neighbors of a node are assigned IDs, then the probability of the node being left out is at most* $P_l \leq 32 \times (k-1) \times S \div (1,000 \times B \times tw_0) + 1 - (1 - BER)^{S \times 8}$*.*

Proof 4 *The node is left out if all the messages of type 1 sent by its neighbors experience collision or a bit error. The probability of loss of each of these messages is bounded by:* $P_c + P_e = 32 \times (k-1) \times S \div (1,000 \times B \times tw_0) + 1 - (1 - BER)^{S \times 8}$. *Since the different collision events are independent, the joint probability is equal to the product of the different probabilities. Henceforth, we obtain:* $P_l \leq (P_c + P_e)^k \leq P_c + P_e = 32 \times (k-1) \times S \div (1,000 \times B \times tw_0) + 1 - (1 - BER)^{S \times 8}$.

Again, this probability can be controlled through the parameter tw_0. For a numerical example, if we assume that all the parameters have the same values as in the numerical example given for Lemma 3, then the upper bound on P_l is 14.79%. Again, this a very conservative upper bound that is almost never reached. For instance, in all our simulation scenarios, the percentage of nodes left without ID is always less than 8% as shown in Figures 20, 21 and 22.

A performance measure of the algorithm is the amount of energy consumed per node during the execution of the algorithm. Since the processing energy is negligible compared to the communication energy, we are taking into account only the latter. The following lemma bounds the average communication energy consumption per node.

Lemma 5 *If on average each node has d neighbors and the average message size in the network is* S_a *bits, then the average communication energy consumption is bounded by* $E_a \leq 15 \times S_a \times (E_t + d \times E_r)$, *where* E_t *and* E_r *are respectively energy consumption per bit for transmission and reception, with probability* $P_e \geq (1 - P_2)^7$, *where* P_2 *is as defined in lemma 3.*

Proof 5 *The proof of this lemma uses the fact that each message is transmitted successfully within two trials with a high probability* P_2 *as proved in lemma 3. We also assume that the the probability of two children nodes choosing the same integer ID (in Phase 1) is negligible since these IDs are randomly chosen from a large set. In*

this case every node causes the sending of 8 different messages: messages of all types except the reinitialization message. All of these messages potentially require resending except the initialization message. Therefore, the number of messages per node is less than or equal to 15, with high probability $P_e \geq (1 - P_2)^7$. *Since every message is a broadcast, we have every node in the neighborhood receiving the message. A message is, consequently, transmitted by 1 node and received on average by d nodes. Therefore, the average consumption per node is bounded by:* $E_a \leq 15 \times S_a \times (E_t + d \times E_r)$.

3.5 Simulation Results

In this section, we show the performance of the algorithm under different simulation settings. We study the effect of different parameters on the performance. In particular, we study the effect of the network size, network density, bit error rate (BER) and the initial value of $timeWait$ on the execution time, the percentage of nodes assigned an ID at the end of the algorithm, and the probability of a message being retransmitted.

Simulations are performed using *GTSNetS*. Nodes are distributed in an equidistant fashion in a square region with the initiator located at the center of the region. The distance between two successive nodes is fixed at 20 meters. The network density is changed by modifying the transmission range: transmission range of 21 meters for a density of 4, 30 meters for a density of 8 and so on. Messages exchange is performed entirely using broadcasts. Channel sensing is performed before sending a message, which reduces the collision probability. Under each setting, each simulation was run 10 times. An average for these 10 runs is used as the final result. It must be noted that the observed variance from run to run for the 10 runs is very low. For example, for example for the case of a network of 1,000 nodes with a density of 4 and BER of 10^{-4}, the variance on the execution time is 0.0014 (with a mean of 5.81 minutes. The variance on the percentage of nodes assigned an ID at the end of

the algorithm is 0.20 (with a mean of 99.7 %) and the variance for the retransmission rate is 0.00098 (with a mean of 7.57 %). The low variance has been observed for all the simulation scenarios.

Figure 17 plots the execution time of the algorithm as a function of the network size while maintaining a fixed network density of 4 neighbors and a typical sensor node bit error rate of 10^{-4} [8]. By the execution time, we do not mean how long the simulation takes to complete. Rather, the execution is a simulation measure of how long the algorithm would have taken to run on a real sensor network with the simulated configuration. It must be noted here that the algorithm execution time is dominated by the $timeWait$ parameter. In particular, it is clear that the time wait duration ranging from 1.5 to several seconds is more than enough for the node to carry any message processing. The communication time is taken into account by the simulator in the computation of the execution time. The simulation scenarios in this section assume that nodes have a transmission rate of $250kbps$, which is the rate for MICAz sensor nodes. As expected, execution time increases with the network size, but remains relatively short even in the case of long initial $timeWait$ value (less than 30 minutes for a network of 10,000 nodes).

Figure 18 plots the execution time of the algorithm as a function of the network density for a network of 1,000 nodes and a bit error rate of 10^{-4}. We can see that the execution time decreases as the density increases. This is due to the fact that density is increased by increasing the communication range, which reduces the number of hops between the initiator and the leaf nodes. The execution time remains relatively short even for a network of low density.

Figure 19 gives the execution time as a function of the bit error rate for a network of 1,000 nodes and a density of of 4. Compared to the network size and density, the bit error rate has a negligible effect on the execution time.

Figures 17, 18 and 19 show that the execution time increases with the initial value

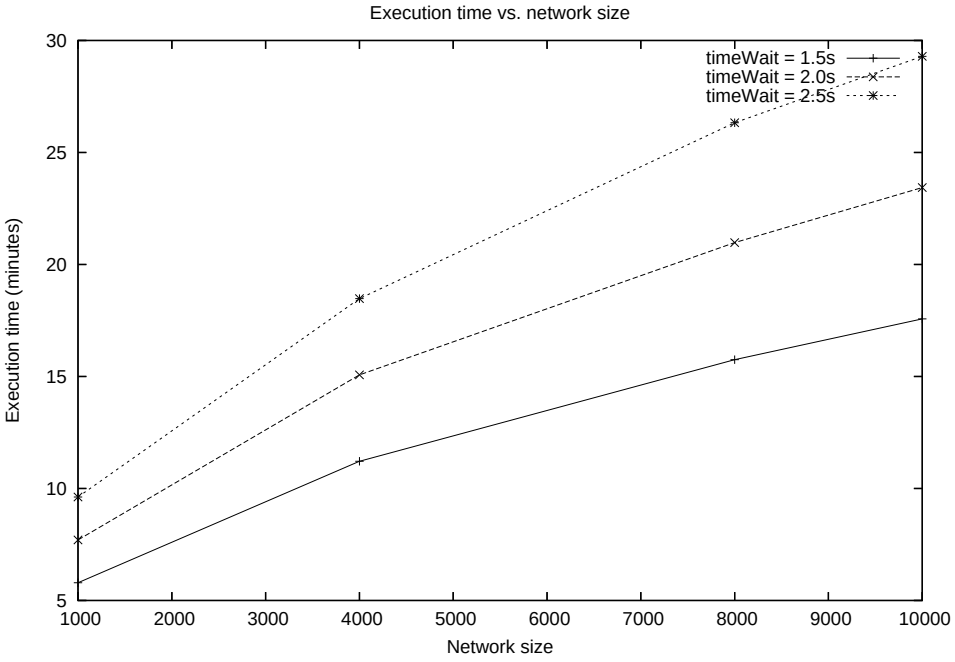

Figure 17: Execution time vs. network size

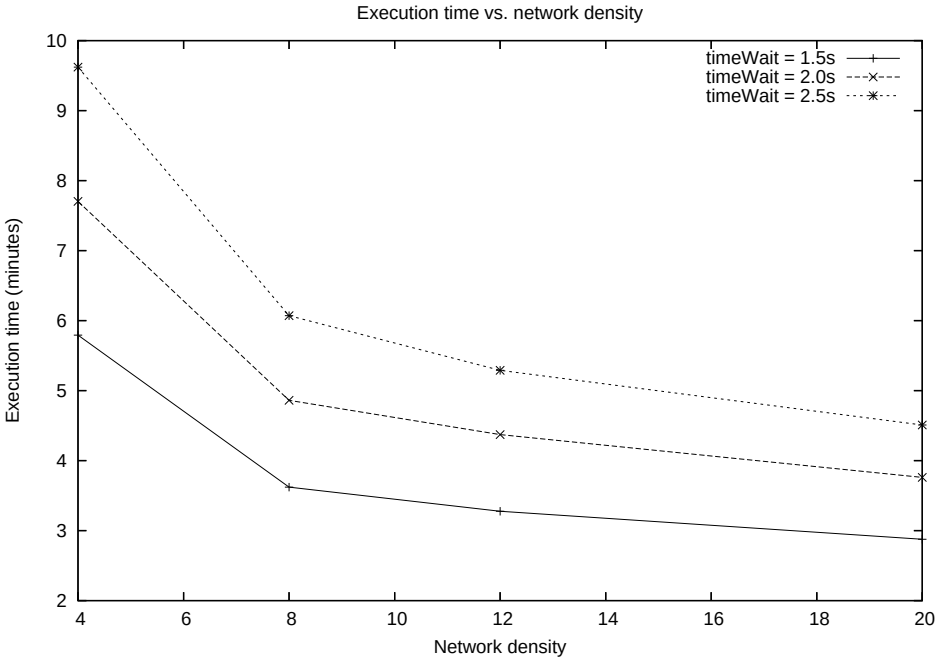

Figure 18: Execution time vs. network density

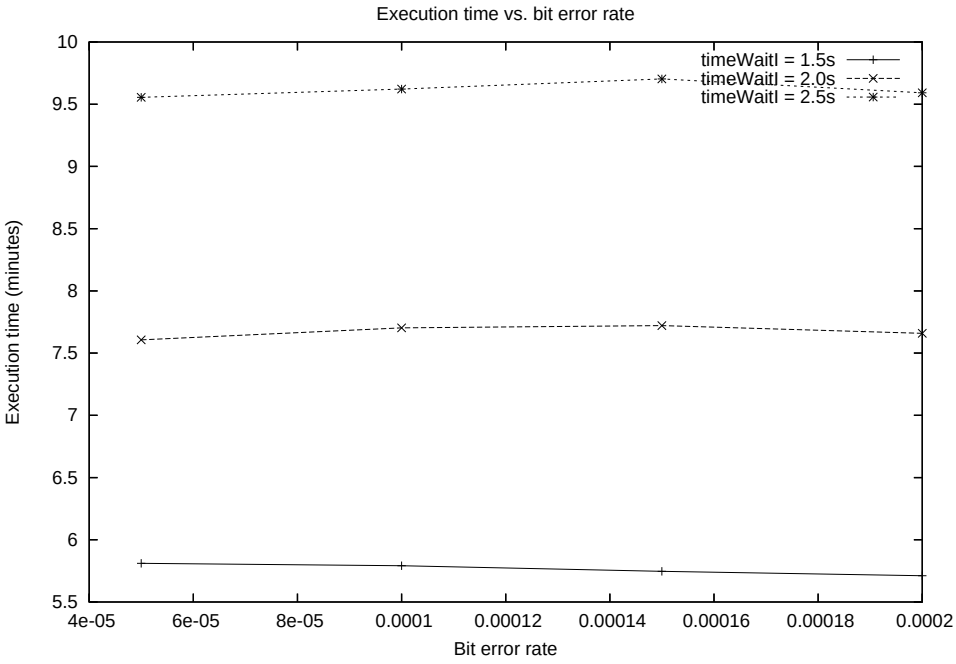

Figure 19: Execution time vs. bit error rate

of $timeWait$. This is expected since nodes wait longer before resending lost messages and before forwarding the initialization messages. This makes the overall algorithm take more time to terminate. It is, therefore, desirable to keep the initial value of $timeWait$ as low as possible, within an acceptable ID assignment percentage.

Figure 20 plots the percentage of nodes assigned a unique ID at the end of the algorithm as a function of the network size with a network density of 4 neighbors and a BER of 10^{-4}. We can see that this probability decreases as the size of the network increases. We can also see that for a specific network size, we can obtain a very high percentage of ID assignments by increasing the initial value of $timeWait$ to a high enough value. However, as this value increases the execution time also increases. With a $timeWait$ initially of 2.5 seconds, we can obtain a percentage of more than 98% even for a large network of 10,000 nodes.

Figure 21 plots the percentage of nodes assigned a unique ID at the end of the algorithm as a function of the network density for a network of 1,000 nodes and a

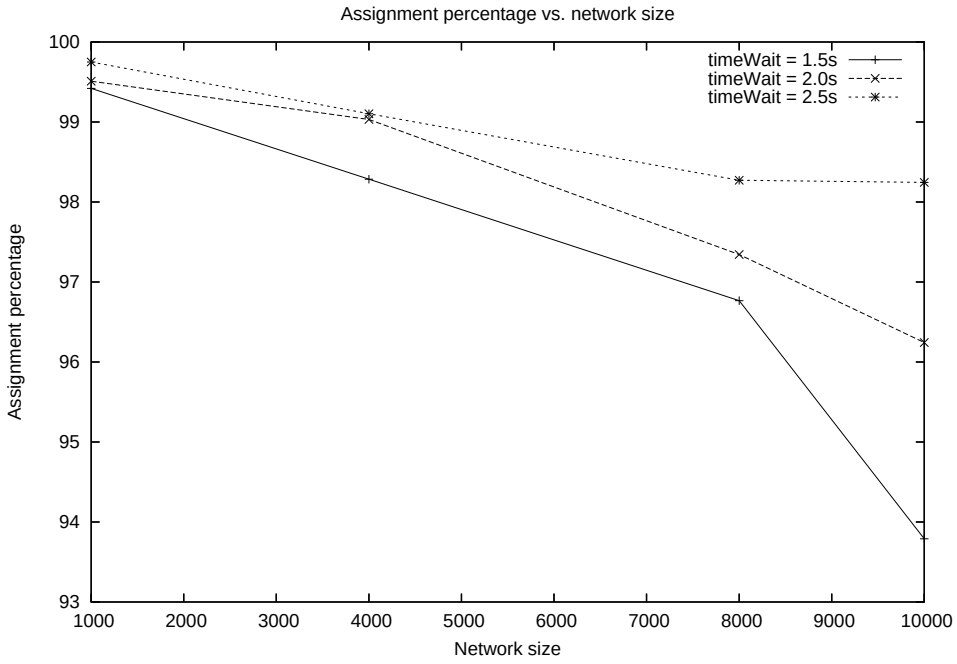

Figure 20: Assignment percentage vs. network size

BER of 10^{-4}. We can see that the percentage of nodes with an assigned ID at the end of the algorithm decreases as the density increases. This is due to the fact that higher density increases the probability of collisions, which reduces the probability of successful reception of messages of type 1 even though more messages are sent in each neighborhood. Messages of type 1 are not retransmitted, and their loss reduces the probability of a node participating in the algorithm. However, it can be seen that this reduction can be balanced by increasing the value of the $timeWait$ parameter.

Figure 22 plots the percentage of nodes assigned a unique ID at the end of the algorithm as a function of the bit error rate for a network of 1,000 nodes and a density of 4 neighbors. We can see that the percentage of nodes with an assigned ID at the end of the algorithm decreases as the BER increases. This is due to the fact that higher BER reduces the probability of successful reception of messages of type 1. Again, the effect of BER is less significant than the effect of network size and density.

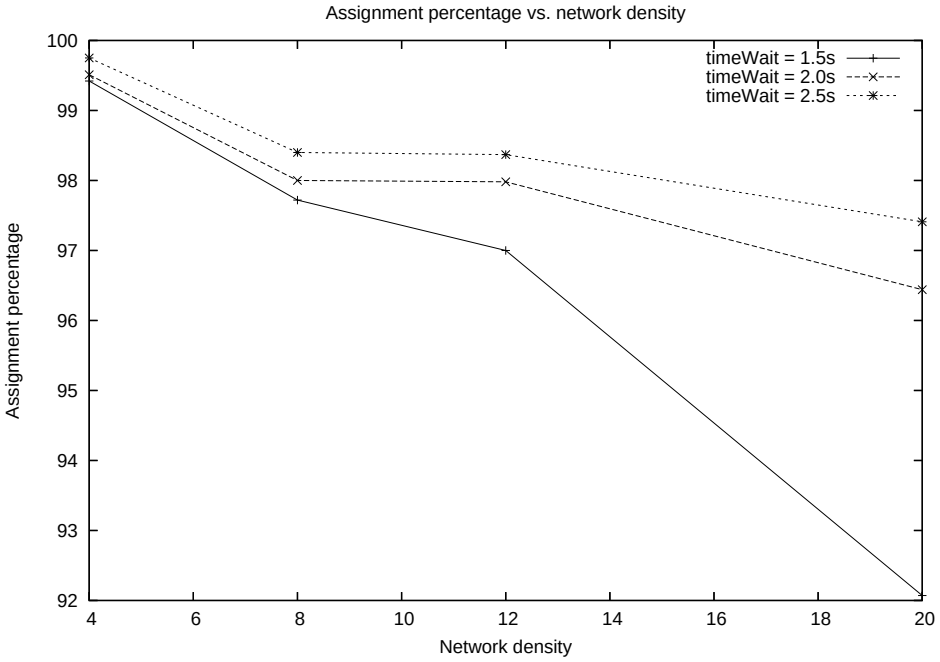

Figure 21: Assignment percentage vs. network density

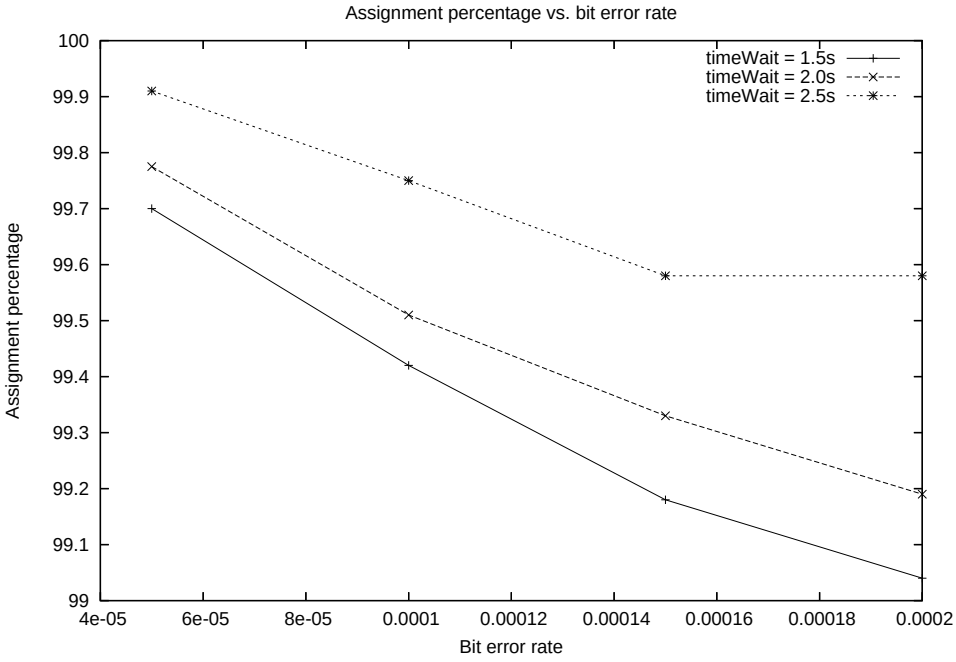

Figure 22: Assignment percentage vs. bit error rate

Figures 20, 21 and 22 indicate that we can increase the percentage of nodes participation in the algorithm by increasing the initial value of $timeWait$. However, such an increase causes the execution time to increase which is not desirable. Thus, there is a tradeoff between the percentage of assigned IDs and the execution time.

Finally, we study the percentage of message loss under various simulation settings. Figure 23 plots the percentage of messages being retransmitted because of a loss as a function of the network size. The network density and the BER are fixed, respectively, at 4 and 10^{-4}. We can see that this percentage increases with size. This is due to the fact that messages are longer on average since more nodes are located many hops away from the initiator. Longer messages take more time to transmit, which makes the occurrence of a collision more likely. As expected, the probability of loss diminishes, when the value of $timeWait$ increases.

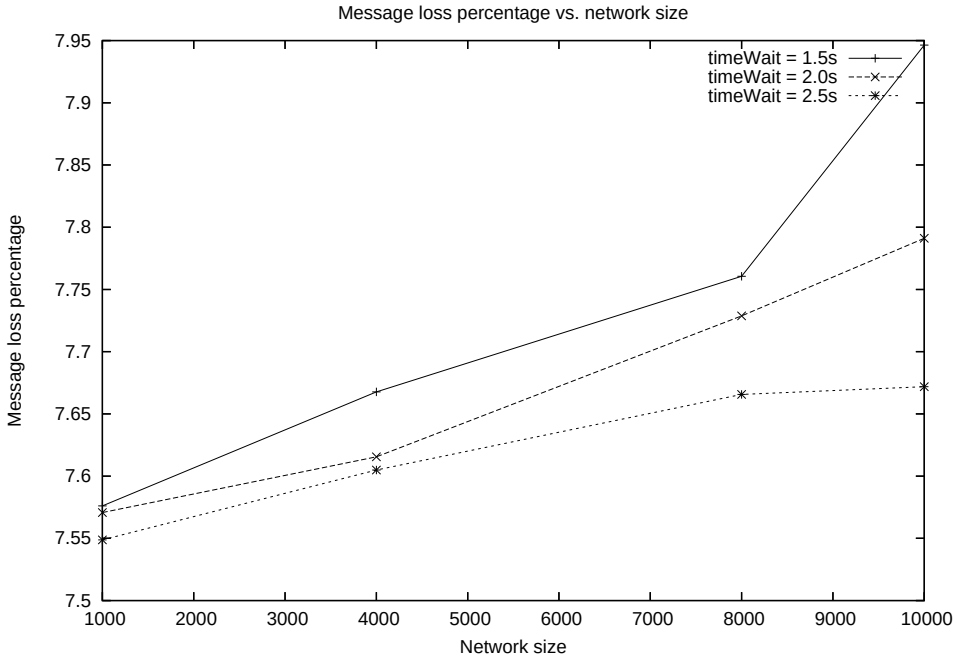

Figure 23: Message loss percentage vs. network size

Figure 24 plots the message loss percentage as a function of network density for a network of 1,000 nodes and a BER of 10^{-4}. We can see that a message loss is more

likely in a network with higher density. This is not surprising since more nodes are competing for each channel, which increase the probability of collision. We can also see that the loss probability can be controlled by increasing the value of $timeWait$.

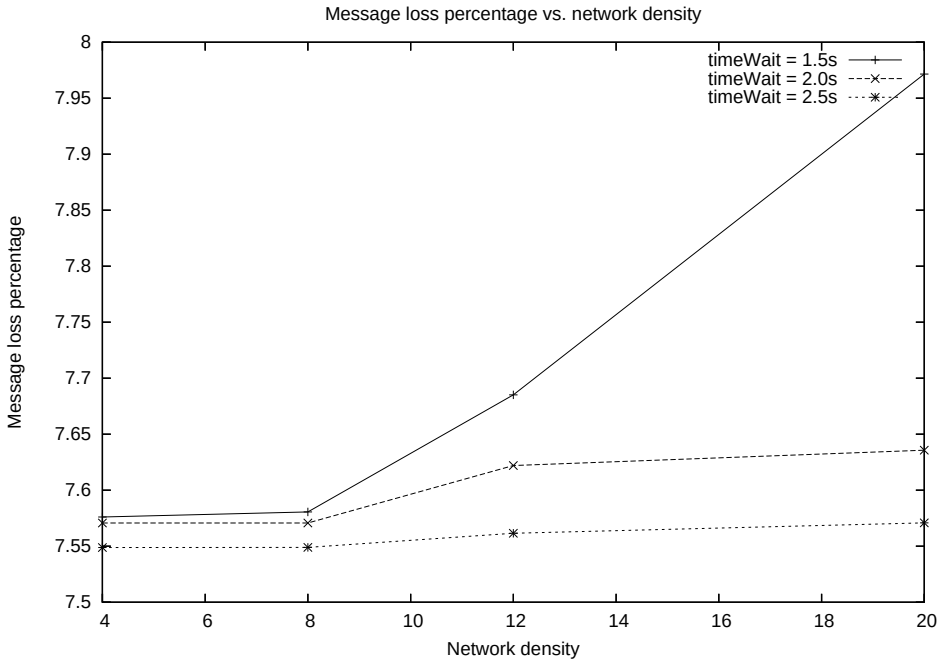

Figure 24: Message loss percentage vs. network density

Figure 25 gives the message loss percentage as a function of bit error rate for a network of 1,000 and a density of 4 neighbors. As expected, a message is less likely to be properly received during the first transmission (higher loss percentage) when the value of the BER is higher.

Based on the results, we can state that the initial value of $timeWait$ plays a central role in the algorithm. It needs to be set appropriately so as to maximize the probability of nodes being assigned an ID at the and of the algorithm and minimize the message loss probability while keeping the execution time under control.

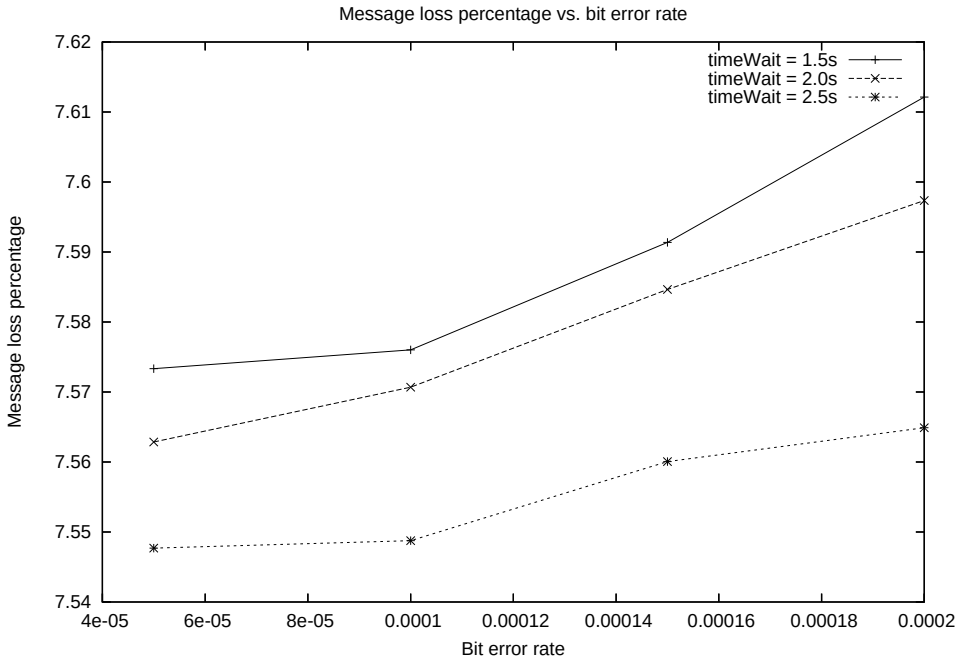

Figure 25: Message loss percentage vs. bit error rate

3.6 Case of Asynchronous Wake-Up

In this section, we extend the unique ID assignment algorithm to cover the case of asynchronous wake-up of nodes. Nodes are no longer assumed to all be awake at the beginning of the algorithm. This implies that the initiator can no longer determine the exact size of the network when it starts assigning final IDs, since new nodes can wake-up later during the execution of the algorithm or after its termination.

To estimate the final size of the network (including the nodes that wake-up after the initial deployment phase), a new parameter ($nbSpareIds$) is introduced to allow each node participating in the initial deployment to require a number of spare IDs from its parents. These results in the initiator node receiving size messages that add up to a total size representing the number of nodes initially deployed and their respective spare IDs. The second phase of the algorithm is modified to account of the spare IDs in the computation of sub-tree sizes. Each node initializes its sub-tree size to $1 + nbSpareIds$ instead of 1 and adds the different sub-tree sizes received from its

children nodes.

When a node receives its final ID, it knows that it has a number of IDs equal to its sub-tree size to accommodate its needs and the needs of its children nodes. In particular, the node updates its ID and stores a list of spare IDs to use when new neighbors wake-up and request an ID. The spare ID list keeps track of which IDs have been assigned and to which nodes they have been assigned.

A new phase is added to assign an ID to a node that joins the network asynchronously. When a node joins the network, it chooses a random 4-byte temporary ID and sends a message requesting a unique ID. Any neighbors that still has one or several spare ID responds with a spare ID assignment message and temporarily marks the corresponding spare ID as assigned. The requesting node chooses the nodes from which it received the first spare ID assignment message as its parent and responds with a confirmation message. When the parent node receives the confirmation message, it marks the assigned spare ID as assigned and confirmed (permanently assigned). Any other node that over-hears the confirmation message marks the spare ID initially assigned to the requesting node as available (no longer assigned). This ID can now be used for a new ID assignment request. If a new node does not receive a spare ID assignment message as a response to its request, it resends the request after a random waiting time. It is possible that a node becomes active before any of its neighbors is assigned an ID. This can happen, for example, if the node joins the network before the termination of the initial round of the algorithm. In this case, the requesting node does not receive a spare ID assignment message. The node waits for a random time and if no initialization message is received (in this case the node acts as an initially active node), it resends its request for a spare ID. The pseudo-code below summarizes the spare ID assignment phase (phase 4).

The number of spare IDs requested by each node that is active at the beginning of the ID assignment algorithm is node-specific. This parameter can vary from node

Algorithm 4 Phase 4: Spare ID Assignment

```
if Newly active node then
    idAssignedF := false
    choose tempId, random 4-byte temporary ID
    choose random time rt
    schedule checking for receiving a spare ID assignment message at rt
    send a spare ID assignment request message
end if
if receive msg a spare ID assignment request message then
    if idAssignedF is true and no corresponding assigned spare ID then
        if Available spare ID then
            mark available spare ID as assigned
            choose random time rt
            schedule checking for confirmation at rt
            send a spare ID assignment message
        end if
    end if
    if idAssignedF is true and msg.tempId corresponds to an assigned spare ID then
        choose random time rt
        schedule checking for confirmation at rt
        resend a spare ID assignment message
    end if
end if
if receive msg a spare ID assignment message then
    if idAssignedF is true then
        resend a spare ID confirmation message
    end if
    if idAssignedF is false then
        myId := msg.spareId
        idAssignedF := true
        send a spare ID assignment confirmation message
    end if
end if
if receive msg a spare ID assignment confirmation message then
    if myId == msg.dst then
        mark corresponding spare ID as assigned and confirmed
    end if
end if
```

to node depending on the anticipated network deployment patterns. Following are examples of different methods to choose appropriate values of the number spare IDs per node.

1. All nodes request the same number of spare IDs: This simple method can be used when the network is expected to have a uniform density during the initial deployment phase and after taking into account the nodes waking up after the initial execution of the algorithm. In this case, the same number of nodes is expected to wake-up in the neighborhood of all nodes initially deployed.

2. A node requests a number of spare IDs that is proportional to the number of neighbors it has during the initial deployment phase. This method is used when there are different density levels in various areas of the deployment region. It assumes that a fixed percentage of nodes in all regions participate in the initial deployment phase and uses this percentage to predict the expected number of neighbors per node that will ultimately wake-up and request IDs.

3. A node requests a number of spare IDs that is inversely proportional to the number of neighbors it has during the initial deployment phase. This method is appropriate when the final network density is constant throughout the network. This implies that the number of initial neighbors a node has is inversely proportional to the number of neighbors expected to wake-up and request a spare ID.

In the last two cases, a node needs to know the number of active neighbors it has during the initial deployment phase. This number can be easily determined by counting the number of initialization messages sent in its neighborhood. In fact, each node that participates in the algorithm after the initial deployment phase sends exactly one initialization message.

3.7 Simulation Results for the Asynchronous Wake-Up Case

In this subsection, we use the same network configuration used to study the ID assignment algorithm in the case of synchronous wake-up. The only difference is that nodes may not all be awake at the beginning of the algorithm. Each node is randomly configured to be initially awake or not. The asynchronous wake-up probability, *pAsynchronous*, is the probability of the node not being awake during the initial phase of the algorithm. All the simulations data was collected by averaging values obtained from 10 different runs.

3.7.1 Effects of the Asynchronous Wake-Up

Here, we try to assess the effect of asynchronous wake-up on the overall performance of the ID assignment algorithm measured by the percentage of nodes that end up being assigned a unique ID. This percentage is measured for networks of various sizes, densities and values of *pAsynchronous*. We also look at the effect of parameter values for the different methods used to compute the number of spare IDs requested by each node participating in the initial phase of the algorithm.

Figure 26 plots the assignment percentage (percentage of nodes assigned a unique ID at the end of the algorithm) for networks of various sizes. The network density was fixed at 20 neighbors per node and the initial value of the $timeWait$ parameter was set to 2.5. The asynchronous wake-up probability is fixed at 60%. For comparison, the assignment percentage is also plotted for the synchronized wake-up case. In both cases, the assignment percentage decreases as the network size increases. The main difference consists of the higher assignment percentage values for the asynchronous wake-up case. This due to the fact the lower collision rate during the phase of the algorithm involving the nodes initially awake in the case of asynchronous wake-up. In fact, only 40% of the nodes participate in the initial phase of the algorithm, which

results in a lower network density and therefore, lower collision rate and higher assignment percentage.

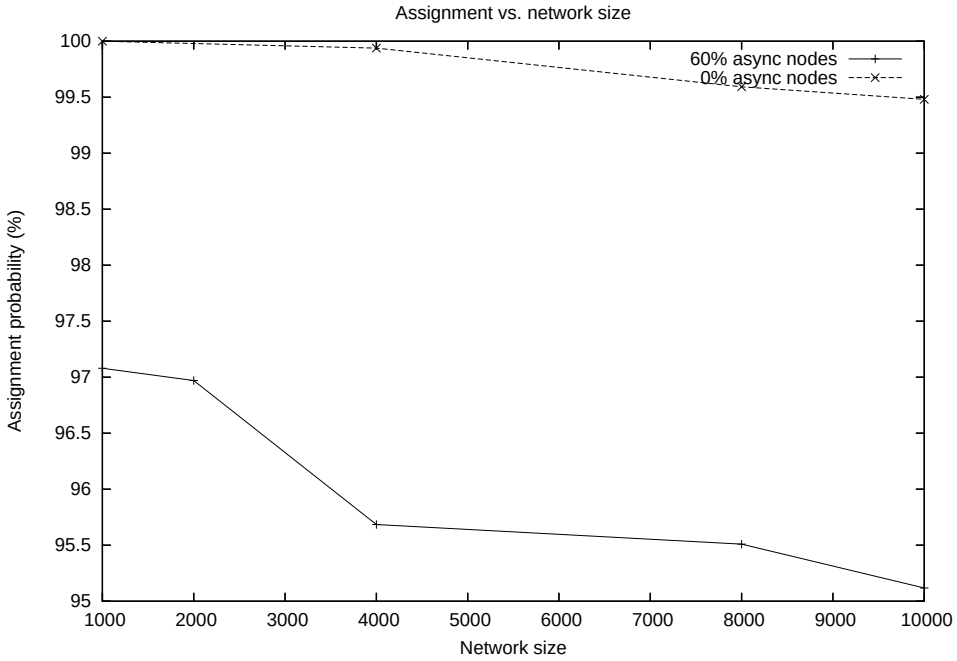

Figure 26: Assignment percentage vs. network size with asynchronous wake-up

Figure 27 gives the assignment percentage for various network densities. Here, the network consists of $1,000$ nodes with an initial $timeWait$ value of 2.5 seconds. Each node participating in the initial phase of the algorithm requests 4 spare IDs. The data is plotted for two different values of the asynchronous wake-up probability, in addition to the synchronized wake-up case. A threshold effect can be seen from the case of 60% asynchronous wake-up. In fact, for networks of low densities, having only 40% of the nodes initially awake results in networks with very low density during the initial phase. This in turn can result in disconnected networks and potentially leaving without an assigned ID a large portion of the nodes initially awake and their asynchronous neighbors that try to obtain an ID upon waking-up. A high assignment probability is obtained when the percentage of nodes initially awake is sufficient to maintain a high connectivity as is the case for high density values and/or low asynchronous wake-up

probability.

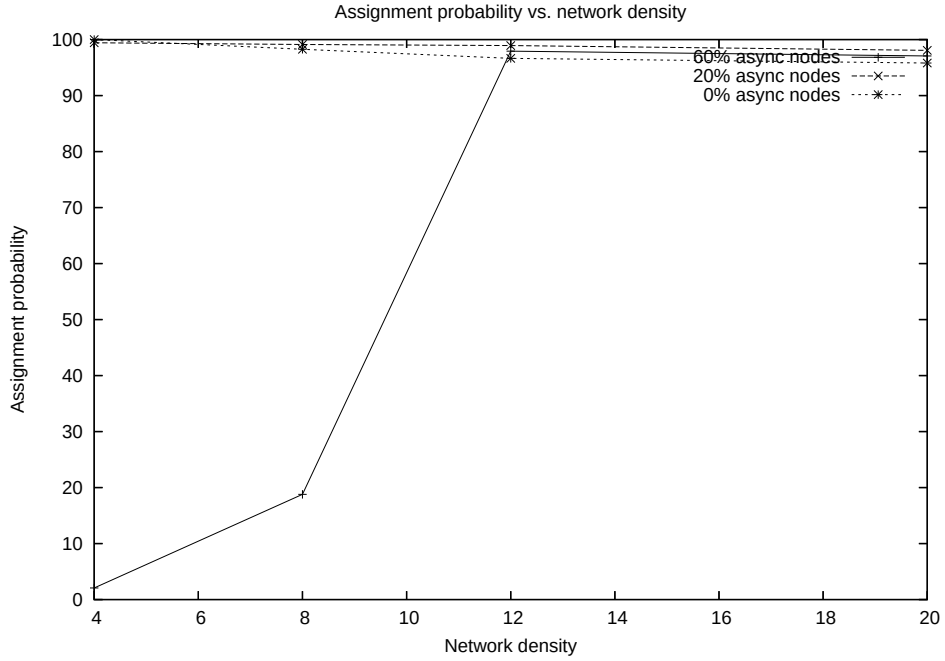

Figure 27: Assignment percentage vs. network density with asynchronous wake-up

Figure 28 gives the assignment percentage as a function of the percentage of nodes not participating in the initial phase of algorithm. The network size is maintained at $1,000$ nodes and the network density is 20 neighbors per node. Each node requests 4 spare IDs during the initial phase of the algorithm. Again, we can see that the assignment percentage increases as the asynchronous wake-up probability increases for moderate $pAsynchronous$ values. When too few nodes are awake during the initial phase (high $pAsynchronous$ values), the network density becomes too low during the initial phase, which results in high collision rates and low assignment probability.

3.7.2 The Effects of the Number of Spare IDs and Its Method of Allocation

Simulations were also conducted to illustrate the importance of the number of spare IDs that each node initially requests for use when neighboring nodes joining the network after the completion of the initial phase of the algorithm. We simulate a

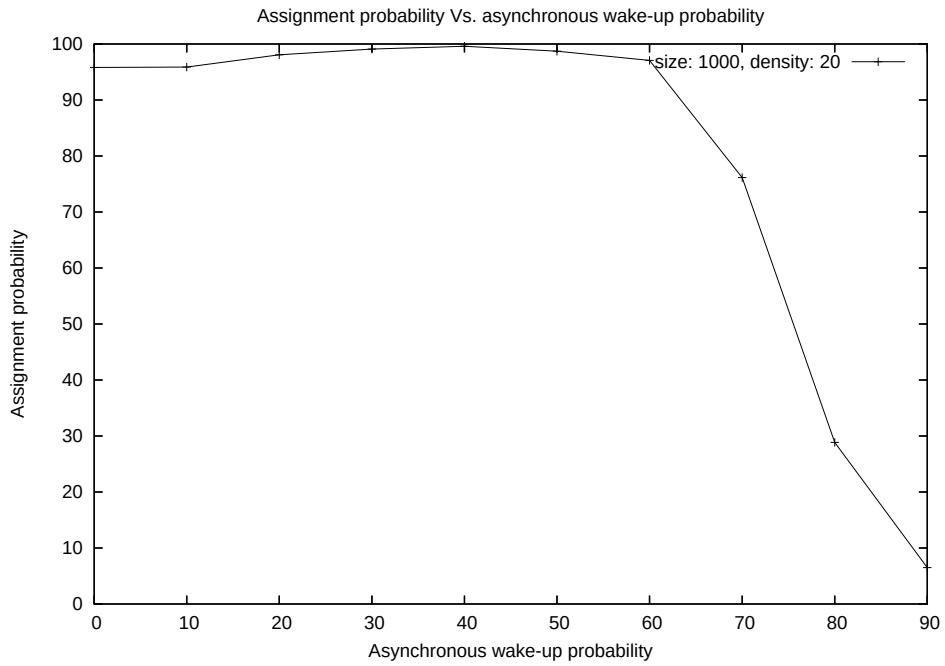

Figure 28: Assignment percentage vs. asynchronous wake-up probability

network of 1,000 nodes and a density of 20 neighbors. The network is split into two halves with different values of the *pAsynchrounous* parameter. In the first half, each node has probability of 40% not participating in the initial phase of the algorithm. This probability is 60% in the second half. Two metrcis are of importance here. The most important metric is the percentage of nodes in the network that end up receiving an ID. An other important metric is how efficient this assignment was accomplished in terms of the ratio between the number of allocated IDs (including spare IDs) and the number of assigned IDs. Of course, the ideal situation is one where all allocated IDs are needed (assigned). This metric is important since the number of allocated IDs determines the ID size needed to code each unique ID.

Table 1 gives the assignment percentage and the ratio of the number of allocated IDs and the number of assigned IDs when using the first method presented in the previous subsection. Here, all nodes request the same number k of spare IDs. The table provides the results as a function of the requested number of spare IDs, k. As

expected, the percentage of nodes successfully assigned an ID increases with the value of the coefficient m. However, the ratio between the number of allocated IDs and the number of IDs assigned to nodes increases. This means that the number of wasted IDs (spare IDs that end up not needed for unique identification of nodes asynchronously joining the network) IDs increases with k, which can result in the number of bits required to represent each ID increasing. It is possible that in certain cases, a lower probability of assignment is preferable to a slightly higher probability that requires larger ID size. For example, changing k from 2 to 4 results in an increase of 60% in the ratio between total number of IDs allocated and the the number of IDs used, while only improving the assignment percentage by less than 5%.

Table 1: Assignment percentage and ratio between allocated and assigned IDs vs. coefficient k when each nodes requests an equal number k of spare IDs

Coefficient k	Assignment percentage (%)	Ratio of allocated and assigned IDs
1	77.82	1.04
2	95.15	1.29
4	99.65	2.04

Table 2 give the results when using the second method, where each node requests a number of spare IDs that is proportional to the of its neighbors initially active. That is the node determines the number of its neighbors by counting the number of initialization message that it receives. It requests a number of spare IDs that is equal to the number of its neighbors multiplied by a parameter m. The table provides the ID assignment percentage and the ratio between the number of allocated and the number of assigned IDs as a function of the parameter m. Again, the percentage of nodes successfully assigned an ID increases with the value of the multiplicative coefficient corresponding to an increase in the number of requested spare IDs per node. At the same time, the algorithm allocates a higher number of unnecessary IDs as the coefficient m increases.

Table 3 illustrates the results when using the third method, where each node

Table 2: Assignment percentage and ratio between allocated and assigned IDs vs. the coefficient m when each node request a number of spare IDs that is proportional to the number of its neighbors

Coefficient m	Assignment percentage (%)	Ratio of allocated and assigned IDs
0.1	77.6	1.05
0.5	90.37	1.47
1	97.45	2.05
1.5	99.02	3.04

requests a number of spare IDs that is inversely proportional to the of its neighbors initially active. In particular, each node requests a number of spare IDs that is equal to the inverse of the number of its neighbors multiplied by a parameter d. The table provides the ID assignment percentage as a function of the coefficient d. Again, the percentage of nodes successfully assigned an ID increases with the value of the number of requested spare IDs per node. As can be seen from the results, this method is ideal for the deployment scenario used here. In fact, since the network density is fixed, it is clear that the number of node neighbors that will wake-up asynchronously and request IDs is inversely proportional to the number of neighbors initially active.

Table 3: Assignment percentage and ratio between allocated and assigned IDs vs. coefficient d when each node requests a number of spare IDs that is inversely proportional to the number of its neighbors

Coefficient d	Assignment percentage (%)	Ratio of allocated and assigned IDs
1	78.57	1.03
5	98.93	1.31
10	99.61	1.97

A way of comparing the performance of the three methods of spare ID allocation it to look at the ratio of the number of allocated IDs and the number of assigned IDs for a given percentage of ID assignment. For example, in the previous deployment strategy we can see that the third method has a ratio of only 1.97 for a percentage assignment of more than 99% while the second method gives a ratio of 3.04 for a similar assignment percentage. This means that for this specific deployment strategy

the best method is the third method where nodes request a number of spare IDs that is inversely proportional to the number of neighbors that are active at the beginning. Indeed, this not surprising since in this case we have two clusters with different values of the probability *pAsynchronous* of nodes participating in the initial round of the algorithm. As the overall density in both clusters is identical, this indicates that the higher the number of neighbors that are active in the beginning, the smaller the number of nodes that will join the network asynchronously and request an ID. We call this deployment scenario a "clustered activation" since the network is composed of clusters with nodes that have different probabilities of being active during the initial deployment phase. Table 4 summarizes the ratio between the number of allocated and assigned IDs for the three methods in the case of an assignment percentage of at least 99%.

Table 4: Ratio between allocated and assigned IDs for different spare ID determination methods for the clustered activation case with a 99 % assignment

Method	Parameter value (k, m or d)	Ratio of allocated and assigned IDs
Equal	4	2.04
Proportional	1.5	3.04
Inversely pr	10	1.97

In contrast with the case of a "clustered activation", a "clustered network" corresponds to the case where different clusters have different densities. We simulate the sensor network described before, except that now all nodes have the same activation probability ($pAsynchronous = 50\%$ but the two clusters have on average different densities. In particular, 10% of the nodes where randomly chosen and removed from the first cluster to be added to the second. The nodes removed from the first cluster are randomly place in the second following a uniform distribution. This situation corresponds to one cluster where nodes have on average a density of 18 neighbors while in the second cluster nodes have an average density of 22 neighbors. The probability of a node being active in the initial round of the ID assignment algorithm is

the same in both clusters. Table 5 give the ratio between the number of allocated and assigned IDs for three methods when the assignment percentage is at least 99%. As can be seen, the natural method for this case where the network is composed of clusters of different densities is the second method where nodes request a number of spare IDs that is proportional to the number of active neighbors in the beginning. This result is not surprising and comes from the fact that the *pAsynchronous* value is fixed across clusters and, therefore, the total number of a node's neighbors that will join the network asynchronously is, on average, proportional to the number of neighbors participating in the initial round of the algorithm.

Table 5: Ratio between allocated and assigned IDs for different spare ID determination methods for the clustered network case with a 99 % assignment

Method	Parameter value (k, m or d)	Ratio of allocated and assigned IDs
Equal	4	1.98
Proportional	1	1.82
Inversely pr	10	2.06

Another scenario is the simple "uniform deployment" where regions of the network have the same density and all nodes have the same probability of being initially active. The results for this scenario are presented in Table 6. No surprisingly, the best method for this scenario is for each node to request an equal number of spare IDs. However, the two other methods still give reasonably low values of the ratio between the number of allocated IDs and the number of assigned IDs. In the previous two scenarios (clustered activation and clustered network), there are two regions of the network with different requirements in terms of number of spare ID per active node. This situation resulted in the parameters (e.g., k for the first method) having to be set to the value corresponding to the value that can satisfy the cluster with the highest requirement in terms of number of spare ID. In contrast, in the case of the uniform deployment there is similar average requirement in terms of the number of spare IDs, and this network-wide requirement is lower compared to the clustered activation and

the clustered network cases.

Table 6: Ratio between allocated and assigned IDs for different spare ID determination methods for the uniform deployment case with a 99 % assignment

Method	Parameter value (k, m or d)	Ratio of allocated and assigned IDs
Eequal	3	1.57
Proportional	0.8	1.73
Inversely pr	8	1.69

3.7.3 Discussion on Setting the Parameter of Spare ID Allocation Methods

The simulation results in the previous subsections demonstrate that our algorithm can assign unique IDs with a good performance level for networks with asynchronous wake-ups as long as enough nodes participate in the initial phase of the algorithm. It is clear that all the methods used to determine the number of spare IDs per active node can result in a good performance as long as the parameter used (e.g., k for the first method) is set properly. In addition, depending on the network topology (e.g., clustered versus uniform) and activation scenario (uniform versus different activation scenarios in different regions), one method or the other is preferable. In the absence of any prior knowledge of the network topology and activation scenario, the first method seems to be the best choice as it generally gives a ratio of the number of allocated IDs to the number of assigned IDs that is close to the one given by the best method.

To correctly set the parameter of the spare ID allocation method, the user can rely on prior knowledge of the network topology and node wake-up patterns. For example, if all that is known is the overall network density in each region and no information is available on nodes wake-up patterns, a good strategy can be to use the inversely proportional allocation method. A conservative approach for each node can be to request a number of spare IDs that is equal to the total number of its neighbors (known from network density) minus the number of its neighbors participating in the initial round of the algorithm. This information is, of course, obtained by counting the

number of initialization messages sent in the node neighborhood. This conservative approach ignores the fact that a node that wakes-up asynchronously might have several neighbors that participate in the initial round of the algorithm. Only one of these neighbors ends up assigning a spare ID to the asynchronous node. This means that nodes do not need to require the maximum number of spare IDs, instead each can require a number of spare IDs that is equal to the estimated number of nodes that will wake-up asynchronously divided by the number of neighbors that are initially active. For example in the case of the "clustered activation" network used in the previous subsection, we know that the final density in the network is 20 neighbors. The probability of a node being active in the beginning is 40% in one of the first cluster and 60% in the second. This means that on average a node in the first cluster has 12 neighbors that will wake-up asynchronously and request a spare ID, while in the second cluster a node has 8 asynchronous neighbors. The parameter for the third method of spare ID allocation should be set to 12 in one cluster and 8 in the second cluster. This gives a network-wide average parameter of 10 neighbors divided among the nodes already awake in the neighborhood. This number corresponds to the result in Table 4.

Another situation is when the final density of the network is not known in advance, but the probability of a node being active in the beginning of the deployment is known. In this case each node can use the second method to request a number of spare IDs that is equal to the number of its active neighbors multiplied by a coefficient. This coefficient can be determined from the known percentage of nodes that are initially. For example, if it is known that 20% of the nodes will be active during the initial phase of the algorithm ($pAsynchronous = 0.20$), then 4 is the value of the coefficient needed to guarantee that there is enough spare IDs to cover the 80% of the nodes that will wake-up asynchronously. Again the number of active neighbors is obtained by counting the number of initialization messages. In the example presented in Table 5

the network is composed of two clusters having different densities (on average 18 and 22 nodes, respectively). On average a node has a probability of 50% of being active in the begining of the algorithm. This means that a coefficient of 1 is sufficient when using the second method of spare ID allocation.

A less likely case is when both the final density and the number of neighbors initially awake are known. In this case, the user knows before the deployment exactly for each node the maximum number of neighbors that will wake-up asynchronously and request a spare ID. In this case, each node can be set to request this number and no information (e.g., number of active neighbors) is needed after the deployment to determine the number of spare IDs needed per node. This case is similar to the "uniform deployment" presented in the previous subsection. The network has a density of 20 neighbors per node with a probability of 50% of a node being active in the first round of the algorithm. Theoretically, each active node only needs to request 1 spare ID since the number of asynchronous nodes is on average equal to the number of active nodes. However, since the exact number of asynchronous nodes is random it is possible that in a specific neighborhood more than half of the nodes wake-up asynchronously and request a spare ID. In Table 6, we see that a minimum of 3 spare IDs per active node is needed to guarantee a high percentage of ID assignment.

3.8 Discussion

As can be seen from the theoretical and simulation results, the proposed ID assignment algorithm has a startup cost in terms of execution time and energy. During the execution of the initial round of the algorithm (for nodes initially active), the network is not being used for its intended final sensing task. For example in the case of of a network of $10,000$ nodes the algorithm can cost about 30 minutes of the network lifetime. This becomes more problematic when the algorithm has to be restarted several times. Such a situation can occur when not enough spare IDs have been

assigned and a large number of asynchronous cannot receive a unique ID. However, the probability of such a situation can be reduced by properly choosing the spare ID allocation method and setting its parameter as discussed in the previous section.

The algorithm cost is compensated by the benefit of having IDs of minimum length that can be used during normal network operations. Having IDs of shortest length reduces the communication energy cost since transmitted packets are smaller, which results in extended lifetime.

An alternative approach is random ID assignment. In this approach, nodes can be deployed and allowed to randomly choose an ID without any need of coordination. Such an approach does not incur a startup cost. However, to avoid the need of coordination between the nodes it is necessary to choose the random IDs in a way that renders the probability of duplicate IDs negligible or provide a protocol to detect and correct duplicate IDs. This can be done by having nodes choose IDs using a large number of bits to be coded [97]. For example for a network of $10,000$ nodes, each node needs to use IDs of 60 bits (derived in the same way as for the example network of 10^6 nodes discussed in [97]) for the probability of duplicate IDs to be sufficiently small. In this case, the benefit of eliminating the startup cost can be outweighed by the increased communication cost coming from the use of large IDs (about 8 bytes each).

To compare our approach with the randomized ID assignment, it is clear that our approach is suitable when unique IDs are used extensively during the regular network operations, e.g., for message source authentication when security is required or for unicast message exchanges. In the $10,000$ network case if we assume a payload of 8 bytes (e.g., the location coordinates of a tracked object), the use of our ID assignment approach can result in a total message size of only 10 bytes (payload plus ID in the header). The randomized approach gives a message size of 16 bytes. This means a potential increase of 60% of node lifetime that can come from the use of our method.

As long as this increase in lifetime is sufficient to compensate for the estimated initial startup cost of 30 minutes, our approach is more cost effective.

The randomized approach is more suitable when the unique IDs are used on rare occasions during the normal network operations. For example if the unique IDs are only needed for rare node maintenance messages, then the network can afford the extra communication cost to avoid the initial startup time of our approach.

3.9 Conclusion

We presented a solution to the global ID assignment problem in sensor networks. Our solution aims at assigning unique IDs to each node using the minimum number of bytes required to code these IDs. This was obtained using a 3-phase approach. In the first phase, temporary long IDs are assigned. These temporary IDs are used in the second phase to reliably determine the exact size of the network and, therefore, the minimum number of bytes to use. In the third phase, final IDs coded using the minimum number of bytes are assigned.

The algorithm allows nodes to join the network and obtain unique IDs after the initial deployment phase. This is performed at the local level by allowing each node initially participating in the algorithm to request several spare IDs to be assigned to neighbors joining after the initial deployment.

We demonstrated that the proposed algorithm can be tailored to obtain excellent results, both in terms of the percentage of participating nodes and the execution time.

CHAPTER IV

DISTRIBUTED FAULT-TOLERANCE FOR EVENT DETECTION USING HETEROGENEOUS WIRELESS SENSOR NETWORKS

Beside ID assignment, another area where new sensor network algorithms are needed is to provide fault-tolerance for event detection using sensor networks. This chapter presents a localized algorithm framework for fault-tolerant event detection. The need for a new solution and the related work are discussed in Section 4.1. Section 4.2 formulates the fault-tolerance problem and describes our approach to solve this problem. Section 4.3 presents a distance-based error model. Section 4.4 presents an error model that accounts for the relative geographical distribution of the decision quorums. Section 4.5 presents simulation evaluation of the proposed approach. Section 4.6 discusses schemes that can be used to allow nodes to learn and continuously update their error rates. Section 4.7 concludes the chapter.

4.1 Motivation and Related Work

One particular sensor network application that has received a growing amount of attention in the recent years is event detection [107, 15, 19, 110, 17]. In such an application, sensor nodes are tasked to determine whether a particular event of interest is occurring in their sensing range. Such an event could be, for example, a volcanic eruption at specific site [110], or the presence of a specific target [17]. An event could be detected from a high value of the sensor reading, for example. Each sensor node first determines if its sensor reading indicates the presence of an event before sending this information to its neighbors or to a sink node. However, in case

of failure the sensor can produce a false positive or a false negative. That is, a high reading indicating an event occurred when it did not or a low reading indicating the absence of event when one did occur.

Event detection is commonly performed using a large number of unreliable low-cost sensor nodes. These nodes can each have a high probability of errors (misses and false-positives). It is, therefore, important to develop fault-tolerant mechanisms that can detect detection faults and take appropriate actions. A possible solution is to provide a high degree of redundancy to compensate for faulty nodes. However, the cost sensitivity and energy limitation of sensor networks make such an approach undesirable [48].

In this environment, collaboration between neighboring nodes can be used to increase the reliability of the detection decisions. This is valid if we assume that failures at neighboring nodes are not correlated, yet there is a spatial correlation between occurrences of detected events at local nodes. In other words, a failure at node n is independent from failures at any of its neighbors. On the other hand, the presence of the detected event (e.g., a chemical agent) at node n is highly correlated with the situation at its neighbors.

Here, we address fault-tolerance in the context of distributed binary detection. A node n is trying to decide whether or not a specific event is present within its coverage range. A binary variable is used to code this decision, with a value of 1 when an event is detected, and a value of 0 otherwise. In its decision scheme, the node uses the sensed data obtained by its local sensor as well as the decisions at its neighboring nodes, assuming spatial correlations.

Distributed fault-tolerance for event detection using the assumption of spatial correlation was first considered in [49]. The algorithm in [49] assumes that all nodes in the network have the same detection error probability and that this rate is known prior to the deployment. These assumptions can be unrealistic. In fact, a node can become

faulty with time either because of a lower energy level or because of aging or unsuitable environmental or operating conditions, thereby changing its error probability. We can also have a heterogeneous sensor network with nodes that have different operational capabilities and accuracy levels. Moreover, the proposed algorithm in [49] is not well suited for highly localized events where the event region is very small. In fact, all nodes within a node communication range are given identical weights in the decision scheme regardless of their distances.

The work in [49] has been followed by two other efforts dealing with the same problem of event region detection. In reference [23], the authors provide comments on the [49] paper and correct some of the mistakes in the theoretic analysis section. In [57], the authors extend the model in [49] to account for the fact that sensor errors have two different sources. An error could be noise-related or coming from a sensor fault. They also discuss the choice of the appropriate neighborhood size. However, they assume again that neighboring nodes of n at any distance have the same accuracy as estimators of the real situation at n. In such a case, the failure probability of the distributed decision scheme can be reduced by increasing the neighborhood size. This assumption may introduce a large number of new errors in the case of a highly localized event. In reference [57] as in [49], it is assumed that all nodes have the same probability of failure and that this probability is known prior to the deployment.

We propose a new approach that considers the case where nodes can have different failure probability values. This allows us to handle various types of failures including noise-related failures, biased measurement, drift over time, stuck-at failures, calibration-related failures, environment-related failures, etc. Our approach can be used as a general distributed fault-tolerance mechanism for any application where nodes may have different accuracy levels. These differences can result from different locations, heterogeneous operating conditions (different sensors, different hardware conditions), different deployment times, etc.

We also consider two new distributed error models that take into account the location and relative position of sensor nodes. The first model takes into account the fact that nodes that are closer to each other have a higher spatial correlation than nodes that are farther apart. The second model accounts for the importance of the relative geographical distributions of the two voting quorums (the two subsets of neighbors deciding the presence of an event or its absence, respectively). In this model, if a node has 50% of its neighbors reporting the same decision (e.g., '1': event detected) there is a difference in terms of the value of such a decision depending on whether these neighbors are geographically distributed around the node or are all on the same side. This comes from the observation that an event detected in all sides of a node is more likely to be present at the node itself than an event that was detected by nodes only on one side of the node.

Finally, we also describe several estimation mechanisms that can be used by sensor nodes to continuously learn their error rates. The proposed mechanisms are based on the moving average and geometric moving average. Both schemes can provide accurate and timely estimation of node error rate when their parameters are set appropriately. Nodes are no longer assumed to know their error rates prior to the deployment. In addition, the proposed learning schemes allow the algorithm to handle the situation where node error rate changes over time.

4.2 The Distributed Fault-Tolerance Solution with Different Probabilities of Failure

In this section we describe our solution used to provide fault-tolerance for distributed event detection while taking into account the possibility of nodes having different accuracy levels. The accuracy levels can differ for several reasons:

1. Nodes may have heterogeneous sensors with different quality levels. This leads to different probability of failures (miss probability and false alarm probability)

at different nodes.

2. Some nodes may suffer degradations during the deployment process. For example, in the case of a forest where the sensor network is deployed by dropping nodes from the air, nodes may suffer different impact effects degrading the quality of the sensor readings. This can result in different detection failure probabilities between neighboring nodes.

3. Failure probabilities may be a function of the node distance from the detected object. In this case, if a node n is implementing fault-tolerance using the correlation between its local decision and those of its neighbors, it is possible that the neighbors closer to n give a more accurate estimate (lower probability of failure) of the real situation at n than would the neighbors located farther from n.

4. Failure probabilities may be function of the sensor age. In this case, the sensor performance degrades over time and the sensor may become biased or suffer a gradual drift. The sensor can also remain stuck at the same value independently of the event reality. If using a reconfiguration mechanism, such as the one in [109], nodes that have been active for different periods of time will have different failure probability levels.

5. Node accuracy may be affected differently in the presence of changing environment conditions such temperature, rain, snow, etc.

In this section, we assume that a node n has a way of learning, through estimation, its own failure probability and sharing it with its neighboring nodes. Methods to estimate these probabilities are presented in Section 4.6. Further, our solution does not assume a specific probability distribution of the faulty sensor readings such as Gaussian as is assumed in [49, 57]. We also do not assume that the accuracy level of

a specific node remains constant over time or that all nodes have the same accuracy levels. Relaxing these assumptions makes our solution more robust and enables it to handle all sources of failures as long as nodes in a specific region are not all faulty.

4.2.1 Problem Formulation

We consider a sensor network composed of N nodes distributed over a detection field. Each node has a sensor and is tasked to detect the presence of a specific event. This decision is made using the node's sensor reading compared to a fixed threshold. For simplification, we consider that the presence of an event corresponds to a high sensor reading, while a low reading indicates its absence. An error occurs when a high reading is reported in the absence of an event (false positive) or when a low reading is obtained even though an event is occurring (detection miss). Errors could be due to noisy measurements or a faulty sensor [57]. Here, we treat errors as a single group regardless of the error origin.

Consider that the mean value of the sensor reading in the presence of an event is m_e, while in the absence of event the mean value is m_n. A reasonable threshold value is given by

$$th = \frac{m_e + m_n}{2} \tag{19}$$

We define the following three binary variables, similar to the ones in [49].

- $T_n(t)$: indicates the actual state at the node n and time t (presence or not of an event).
- $S_n(t)$: indicates the state as obtained from the sensor reading of node n. It could be wrong in the case of failure.
- $R_n(t)$: gives an estimate of the real value of $T_n(t)$ using the $S(t)$ values of the node and its neighbors.

The probability of detection error p_n, at node n, is given by:

$$p_n = P(S_n(t) = a | T_n(t) \neq a) \tag{20}$$

For example, if we assume a Gaussian error term (e.g., noise-related error) with a mean of 0 and variance of σ_n^2 at node n, then the probability of detection error is given by:

$$p_n = Q(\frac{m_e - m_n}{2\sigma_n}) \tag{21}$$

where Q is the tail probability of the Gaussian distribution.

The problem at hand is to define an estimator of the actual state at node n that minimizes the detection error probability. This estimator takes into account the local decision obtained from the node sensor reading as well as the local decisions of the neighboring nodes. We consider a sensor network, $Ne = \{1, 2, 3, ..., N-1, N\}$ containing N sensor nodes. The nodes taken into account by a node $n \in Ne$ in its decision mechanism are all nodes within a fixed range, r. This fault-tolerance range, r, should be fixed so as to minimize the probability of error while keeping the communication energy cost low and taking into account the expected size of the event region. Below, we give a formal definition of this neighborhood.

Definition 1 *We define the fault-tolerance neighborhood (FTN_n) as the set of nodes that a node $n \in Ne$ takes into account in its fault-tolerance decision mechanism. If we consider a fault-tolerance range of r, this neighborhood is given by: $FTN_n = \{ne \in Ne : d(n, ne) \leq r\}$, where $d(n, ne)$ is the Euclidean distance between the nodes n and ne. This set contains the node n itself.*

Below, we define the decision vector taken into account by a node n in its fault-tolerant mechanism. We also define the probability of distributed detection error.

Definition 2 *The estimation fault-tolerance vector (FTV_n) is defined as the vector containing all the $S_n(t)$ of n and the $S_j(t)$ of all its fault-tolerance neighbors. FTV_n*

contains K elements, where K is the number of elements in FTN_n. We have that $FTV_n^k(t) = S_m(t)$, where m is the k^th element of FTN_n. This set contains, in particular the value $S_n(t)$ since $n \in FTN_n$.

Definition 3 *The probability of estimator detection error at node n and time t is defined as:*

$$Pe_n(t) = P(R_n(t) \neq T_n(t)|T_n(t), FTV_n) \tag{22}$$

Using these definitions, the problem at hand consists of finding an estimation function that takes as an input the vector FTV_n and gives as an output R_n that minimizes the probability of error Pe_n.

4.2.2 Optimal Estimator for Fault-Tolerant Distributed Detection

To develop an optimal estimation function, we use the likelihood test ratio (LRT) [105]; that is, to choose $R_n(t) = j$ where $j \in \{0, 1\}$ that maximizes the probability $P(T_n(t) = j|FTV_n(t))$.

Below, we define the power set containing all possible sets composed of any subset of the neighbors of node n and the node n itself.

Definition 4 *We define P_n as the set of all possible FTV_n vectors. P_n can be represented by the power set of the set FTN_n, where a value of 1 in the k^{th} position of a vector $v \in P_n$ indicates the presence of the corresponding node (the k^{th} node in FTN_n) in the subset.*

Define the parameter $p(t)$ as the probability of the true situation being the presence of an event at time t.

$$\begin{aligned} P(T_n(t) = 1) &= p(t) \\ P(T_n(t) = 0) &= 1 - p(t) \end{aligned} \tag{23}$$

Next, we define the following two functions on the set P_n.

Definition 5 *The following two functions F_n^0 and F_n^1 are defined as follows:*

$$\begin{aligned} F_n^j &: P_n \rightarrow \mathbb{R}^+, j \in \{0,1\} \\ F_n^j(v) &= P(T_n(t)=j) \prod_{k|v(k)=j} \frac{1-p_{FTN_n(k)}}{p_{FTN_n(k)}} \end{aligned} \tag{24}$$

where $v \in P_n$ is the current value of the FTV_n vector, and $v(k)$ and $FTN_n(k)$ give the k^{th} elements of the vectors v and FTN_n, respectively. Note that $FTN_n(k)$ corresponds to the k^{th} node in the FTN_n set consisting n and its neighbors and $p_{FTN_n(k)}$ is the local detection error probability of this node defined in equation 20.

The function F_0 represents the product of the elements $\frac{1-p_k}{p_k}$ for all nodes reporting a local decision of 0 in the set containing n and its neighbors multiplied by the probability of non-occurrence of the event. The function F_1 gives the same product for nodes reporting a local decision of 1. We can now define the optimal estimator that minimizes the probability of detection error. This estimator is given in the following definition.

Definition 6 *We define the following fault-tolerant estimator for distributed detection (FTEDD) as an estimator that declares $R_n(t)=0$ if and only if $F_0(FTV_n(t)) < F_1(FTV_n(t))$. The estimator declares $R_n(t)=1$, otherwise.*

The optimality of this estimator is proven in the next theorem.

Theorem 1 *The FTEDD estimator is optimal with respect to the maximum a posteriori (MAP) criterion.*

Proof 6 *For the two possible hypotheses, $T_n(t)=0$ and $T_n(t)=1$, the conditional probability given $FTV_n(t)$ can be obtained using the Bayes' rule. The two posterior probabilities are given by:*

$$P(T_n(t)=j|FTV_n(t)) = \frac{P(FTV_n(t)|T_n(t)=j)P(T_n(t)=j)}{P(FTV_n(t))} \tag{25}$$

where $j \in \{0,1\}$. *The value of* $P(FTV_n(t))$, *the probability of occurrence of the current value of* $FTV_n(t)$ *is given by:*

$$P(FTV_n(t)) = \sum_{j=0}^{1} P(FTV_n(t)|T_n(t)=j)P(T_n(t)=j) \tag{26}$$

We have that the local detection decision for a node $k \in FTN_n$ *is correct with a probability of* $1-p_k$. *Using this information and the fact that the events of errors in the local decisions are independent, it is clear that when the real situation is* $T_n(t)=j$, *the nodes reporting a local decision of* $S_k(t)=j$ *are correct, while the others are faulty. This gives the following conditional probabilities:*

$$\begin{aligned} P(FTV_n(t)|T_n(t)=0) &= \prod_{k|k\in FTN_n} p_k^{S_k(t)}(1-p_k)^{1-S_k(t)} \\ P(FTV_n(t)|T_n(t)=1) &= \prod_{k|k\in FTN_n} p_k^{1-S_k(t)}(1-p_k)^{S_k(t)} \end{aligned} \tag{27}$$

These equations multiply the correctness probability $(1-p_k)$ *for nodes reporting the correct value* (j) *by the error probability for nodes reporting the opposite value.*

We can now compute the posteriori probabilities as follows:

$$\begin{aligned} P(T_n(t)=1|FTV_n(t)) &= \frac{P(FTV_n(t)|T_n(t)=1)P(T_n(t)=1)}{P(FTV_n(t))} \\ &= \frac{p(t)\prod_{k|k\in FTN_n} p_k^{1-S_k(t)}(1-p_k)^{S_k(t)}}{P(FTV_n(t))} \end{aligned} \tag{28}$$

and:

$$\begin{aligned} P(T_n(t)=0|FTV_n(t)) &= \frac{P(FTV_n(t)|T_n(t)=0)P(T_n(t)=0)}{P(FTV_n(t))} \\ &= \frac{(1-p(t))\prod_{k|k\in FTN_n} p_k^{S_k(t)}(1-p_k)^{1-S_k(t)}}{P(FTV_n(t))} \end{aligned} \tag{29}$$

We can now compute the likelihood ratio as:

$$\begin{aligned} \gamma &= \frac{P(T_n(t)=0|FTV_n(t))}{P(T_n(t)=1|FTV_n(t))} \\ \gamma &= \frac{(1-p(t))\prod_{k|k\in FTN_n} p_k^{S_k(t)}(1-p_k)^{1-S_k(t)}}{p(t)\prod_{k|k\in FTN_n} p_k^{1-S_k(t)}(1-p_k)^{S_k(t)}} \end{aligned} \tag{30}$$

It can be easily seen that:

$$\gamma = \frac{1-p(t)}{p(t)} \prod_{k|k\in FTN_n} (\frac{p_k}{1-p_k})^{S_k} (\frac{1-p_k}{p_k})^{1-S_k(t)} \tag{31}$$

This equation can be re-written as:

$$\begin{aligned} \gamma &= \frac{1-p(t)}{p(t)} \prod_{k|k\in FTN_n} \frac{(\frac{1-p_k}{p_k})^{1-S_k(t)}}{(\frac{1-p_k}{p_k})^{S_k(t)}} \\ \gamma &= \frac{(1-p(t)) \prod_{k|k\in FTN_n} (\frac{1-p_k}{p_k})^{1-S_k(t)}}{p(t) \prod_{k|k\in FTN_n} (\frac{1-p_k}{p_k})^{S_k(t)}} \end{aligned} \tag{32}$$

Since $1 - S_k(t) = 0$ when $S_k(t) = 1$, the numerator in the previous equation can be written as follows:

$$\begin{aligned} &(1-p(t)) \prod_{k|k\in FTN_n} (\frac{1-p_k}{p_k})^{1-S_k(t)} \\ &= (1-p(t)) \prod_{k|v(k)=0} \frac{1-p_{FTN_n(k)}}{p_{FTN_n(k)}} = F_0(v) \end{aligned} \tag{33}$$

where $v = FTV_n(t)$. Similarly, the denominator corresponds to $F_1(v)$. We can, therefore, re-write γ as:

$$\gamma = \frac{F_0(FTV_n(t))}{F_1(FTV_n(t))} \tag{34}$$

The estimator minimizes the error if it estimates $R_n(t) = 0$ when $\gamma > 1$ [105], which is equivalent to deciding based on the maximum a posteriori (MAP) criterion. This corresponds to the case of $F_0(FTV_n(t)) > F_1(FTV_n(t))$. This completes the proof of the optimality of the FTEDD estimator.

The following corollaries give the detection error probability of the FTEDD estimator and determine whether it is biased or not.

Corollary 1 *The probability of detection error at node n and time t is given by:*

$$\begin{aligned} Pe_n(t) = p_n(t) &\sum_{v\in\Omega_0} \prod_{k|k\in FTN_n} p_k^{1-S_k(t)} (1-p_k)^{S_k(t)} \\ +(1-p_n(t)) &\sum_{v\in\Omega_1} \prod_{k|k\in FTN_n} p_k^{S_k(t)} (1-p_k)^{1-S_k(t)} \end{aligned} \tag{35}$$

where $\Omega_0 = \{v \in P_n : F_0(v) > F_1(v)\}$ and $\Omega_1 = \{v \in P_n : F_1(v) > F_0(v)\}$.

Proof 7 *An error occurs when the estimator decides a value $R_n(t)$ that is different from the real value $T_n(t)$. The probability of detection error is therefore given by:*

$$\begin{aligned} Pe_n(t) &= P(R_n(t) \neq T_n(t) | T_n(t), FTV_n(t)) \\ &= P(T_n(t) = 1) P(R_n(t) = 0 | T_n(t) = 1, FTV_n(t)) \\ &+ P(T_n(t) = 0) P(R_n(t) = 1 | T_n(t) = 0, FTV_n(t)) \end{aligned} \tag{36}$$

If $v = FTV_n(t)$, the conditional probabilities are computed as follows:

$$\begin{aligned} P(R_n(t) = 0 | T_n(t) = 1, v) &= P(F_0(v) > F_1(v) | T_n(t) = 1, v) \\ &= \sum_{v \in \Omega_0} \prod_{k | k \in FTN_n} p_k^{1 - S_k(t)} (1 - p_k)^{S_k(t)} \end{aligned} \tag{37}$$

In a similar way, we have:

$$\begin{aligned} P(R_n(t) = 1 | T_n(t) = 0, v) &= P(F_1(v) > F_0(v) | T_n(t) = 0, v) \\ &= \sum_{v \in \Omega_1} \prod_{k | k \in FTN_n} p_k^{S_k(t)} (1 - p_k)^{1 - S_k(t)} \end{aligned} \tag{38}$$

And since $P(T_n(t) = 1) = p_n(t)$ and $P(T_n(t) = 0) = 1 - p_n(t)$, we obtain the desired result.

Corollary 2 *Suppose that for all nodes $k \in Ne$, the probability of local detection error $0 < p_k < \frac{1}{2}$, then the FTEDD estimator is biased. However, the estimator is asymptotically unbiased.*

Proof 8 *This corollary comes from the observation that because $0 < p_k < \frac{1}{2}$, we will always have a non-null probability of error: $Pe_n(t) = P(R_n(t) \neq T_n(t) | T_n(t), FTV_n(t)) > 0$ for a finite number of neighbors. The expected value of the estimator decision is given by:*

$$\begin{aligned} E(R_n(t) | T_n(t), FTV_n(t)) &= T_n(t) P(R_n(t) = T_n(t) | T_n(t), FTV_n(t)) \\ &+ (1 - T_n(t)) P(R_n(t) \neq T_n(t) | T_n(t), FTV_n(t)) \end{aligned} \tag{39}$$

Since $P(R_n(t) \neq T_n(t) | T_n(t), FTV_n(t)) > 0$, we have that $E(R_n(t) | T_n(t), FTV_n(t)) \neq T_n(t)$. The estimator is, therefore, biased. However, as the number of elements in the

FTN_n increases, the probability of error gets closer to 0 since $p_k < \frac{1}{2}$ for all nodes in the network. In this case, $E(R_n(t)|T_n(t), FTV_n(t))$ approaches the real situation, $T_n(t)$, and the estimator becomes unbiased.

The next theorem allows the expression of the FTEDD estimator as a weighted voting scheme [35] and provides the corresponding node weights. This is in contrast with the majority and $k - out - of - n$ schemes used in [49].

Theorem 2 *When the presence or absence of an event are equally likely ($p_n(t) = \frac{1}{2}$), the FTEDD estimator is equivalent to a weighted voting scheme of the nodes in FTN_n. A node $k \in FTN_n$ has the following weight:*

$$w_k = \ln \frac{1 - p_k}{p_k}, \forall k \in FTN_n \tag{40}$$

Proof 9 *We have that when $1 - p_n(t) = p_n(t) = \frac{1}{2}$, the likelihood ratio is given by:*

$$\gamma = \prod_{k|k \in FTN_n} (\frac{p_k}{1 - p_k})^{S_k} (\frac{1 - p_k}{p_k})^{1 - S_k(t)} \tag{41}$$

Since $\frac{1-p_k}{p_k} = e^{w_k}$, we can write γ in the following form:

$$\begin{aligned} \gamma &= \prod_{k|k \in FTN_n} (\frac{1}{e^{w_k}})^{S_k} (e^{w_k})^{1 - S_k(t)} = \prod_{k|k \in FTN_n} (e^{w_k})^{1 - 2S_k(t)} \\ &= \prod_{k|k \in FTN_n} e^{w_k(1 - 2S_k(t))} = e^{(\sum_{k|k \in FTN_n} w_k(1 - 2S_k(t)))} \end{aligned} \tag{42}$$

It is clear that $\gamma > 1$ is equivalent to $\sum_{k|k \in FTN_n} w_k(1 - 2S_k(t)) > 0$. This sum can be written in the following way by replacing the values of $1 - 2S_k(t)$ with 1 or -1 depending on the values of S_n^k:

$$\begin{aligned} &\sum_{k|k \in FTN_n} w_k(1 - 2S_k(t)) \\ &= \sum_{k \in FTN_n | S_k(t) = 0} w_k - \sum_{k \in FTN_n | S_k(t) = 1} w_k \end{aligned} \tag{43}$$

So $\gamma > 1$ is equivalent to $\sum_{k \in FTN_n | S_k(t) = 0} w_k > \sum_{k \in FTN_n | S_k(t) = 1} w_k$. This corresponds to a weighted majority vote in favor of the hypothesis of $T_n(t) = 0$. This completes the proof of the theorem.

4.3 Distance-Based Error Model for Fault-Tolerant Distributed Detection

In this section, we present a new fault-tolerant event detection scheme that uses a distance-based error model. This scheme allows us to account for the fact that the evidences coming from two different neighbors of n do not necessarily have the same importance when used to estimate the real situation at node n. In fact, the correlation between the real situation at node n and the situation at a node $n_1 \in FTN_n$ is higher than the correlation with the situation n_2 when $d(n, n_1) < d(n, n_2)$. Here, $d(n, n_i)$ is the Euclidean distance between the two nodes.

We use a model inspired by the distance-based signal model in [111]. Consider a node n_k at distance d from an event site. If the true sensor reading of a node collocated at the event site is e_0, then the average sensor reading at n_k can be modeled as:

$$e = \frac{be_0}{d^a} \tag{44}$$

where the parameters a and b represent the attenuation factors and are function of the event propagation characteristics, size of the event region and the deployment terrain properties. Example values are $b = 1$ and $a = 2$ in the absence of obstacles. However, the values of a and b depend on terrain characteristics and propagation properties [111].

To take the neighbor distance into account, we define a new weighted voting model that gives a weight factor to each neighbor that is a function of its relative distance to n compared to other neighbors.

Definition 7 *We define the distance weight wd_k as the weight given to the node $k \in FTN_n$ as follows:*

$$wd_k = 1 + \frac{d(n,k)}{\sum_{m|m \in FTN_n} d(n,m)} \tag{45}$$

And the node weight in the voting scheme is given by:

$$wn_k = wd_k w_k \tag{46}$$

where w_k represents the original node weight, defined previously in theorem 2.

This node is not ideal, since it does not give to each neighbor k a weight corresponding to the exact probability of detection error when using a node $k \in FTN_n$ to estimate the real situation at node n. The computation of this probability requires the assumption that the sensor readings follow a specific probability distribution model, which is not assumed here for the purpose of generality. For example if we assume a Gaussian error term of mean 0 and variance σ_n^2, then the probability of detection error when using a node $k \in FTN_n$ to estimate the real situation at node n is given in the next proposition.

Proposition 1 *Assuming a Gaussian error term, the probability of error when an event occurring at node n is detected by a node $k \in FTN_n$ is given by:*

$$p_k^n = Q(\frac{bm_e}{\sigma_k (d(n,k))^a} - Q^{-1}(p_k)) \tag{47}$$

where m_e is the mean value in the presence of event and $d(n,k)$ is the distance between the two nodes.

Proof 10 *This proposition comes from the assumption that the node k uses the threshold defined in equation 19. In this case an error occurs when an event occurs and is not detected or an event is detected when no event did occur. These two probabilities are equal. Using the mean sensed value in presence of event m_e instead of e_0 in equation 44. The detection error is therefore:*

$$\begin{aligned} p_k^n = Q(\frac{\frac{bm_e}{d(n,k)^a} - th}{\sigma_k}) &= Q(\frac{\frac{bm_e}{d(n,k)^a} - (\frac{m_e - m_n}{2})}{\sigma_k}) \\ &= Q(\frac{\frac{bm_e}{d(n,k)^a}}{\sigma_k} - \frac{m_e - m_n}{2\sigma_k}) \end{aligned} \tag{48}$$

And since $\frac{m_e - m_n}{2\sigma_k} = Q^{-1}(p_k)$, we obtain the result in the proposition.

We note that by using this Gaussian error model, different nodes have different perceived error probabilities for a node n depending on their distances from n. These

probabilities are normally different from the local probability at n. In this scheme, the error probabilities p_k^n can be used instead of the values of p_k by node n to compute each neighbor weight for the decision scheme in definitions 5 and 6 and theorem 2.

4.4 Group-Based Error Model for Fault-Tolerant Distributed Detection

In this section, we develop an error model that takes into account in the fault-tolerance mechanism the geographical distribution of the group of neighbors reporting a specific detection decision. To illustrate this idea, we consider the following examples. We assume, for simplification, that all nodes have the same probability of error p. In the first example, nodes 1, 2, 3 in Figure 29 report a detection decision of 1. In the second example, nodes 2, 4, 6 report the same decision. The idea here is that even though the same number of neighbors of n reported a decision of 1 in the two examples, the second decision is more reliable. This is because if all nodes reporting a decision of 1 are on one side of n it is conceivable that these nodes are at the border of the event region. In this case, the node n is outside of the event region and no event should be detected. On the other hand, if nodes from different sides of n report a decision of 1, it is very likely that an event is also present at n. This is specially true in the case of a convex event region.

To take the geographical distribution into account, we define a new weighted voting model that gives a weight factor to each neighbor that is a function of the geographical distribution of the decision group to which they belong. There are two decision groups $G_0 = \{k \in FTN_n \setminus n : S_k(t) = 0\}$ and $G_1 = \{k \in FTN_n \setminus n : S_k(t) = 1\}$.

Definition 8 *We define the weight wg_j as the weight given to the decision group G_j with $j \in \{0, 1\}$. This group weight is given by:*

$$wg_j = \frac{r - d(n, m_j)}{r} \tag{49}$$

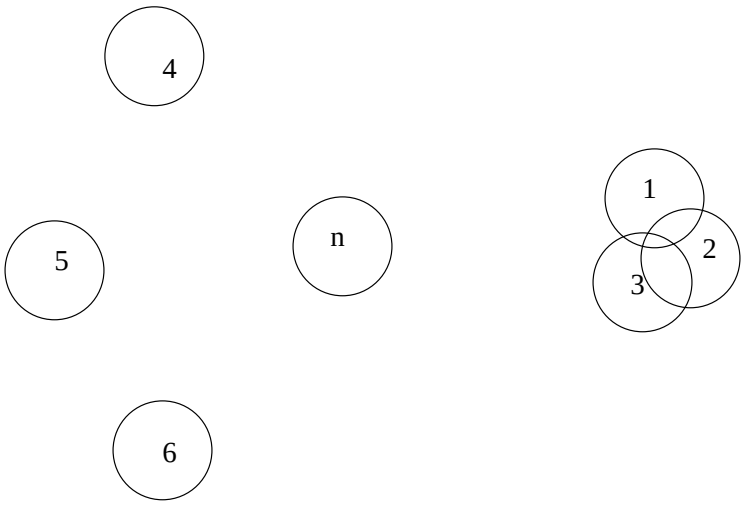

Figure 29: Group-based error model

where m_j is the geographical centroid of nodes in G_j, $d(n, m_j)$ is the distance between n and m_j and r is the fault-tolerance range.

This weight is higher when m_j is closer to n. In fact, the centroid of the group is closer to n when nodes are distributed around n than when the nodes are on the same side while having the same distances from n.

We can now compute the node weight that is a function of both its individual local detection error and its group weight. The new node weight of a neighbor $k \in G_j$ is given by:

$$wn_k = wg_j w_k \tag{50}$$

where w_k represents the original node weight, defined previously in theorem 2.

Using these new weights, we can now compute the probability of detection error when using a node $k \in FTN_n \setminus n$ to estimate the real situation at node n. This probability is given in the following proposition.

Proposition 2 *The probability of error when an event occurring at node n is detected by a node $k \in FTN_n$ is given by:*

$$p_k^n = \frac{1}{1 + e^{wn_k}} \tag{51}$$

Proof 11 *This error probability is obtained by inverting the relationship in theorem 2 and using the new wn_k instead of w_k.*

We can now use these new probabilities of detection errors in the decision scheme instead of the original ones in definitions 5 and 6.

Note that this group-based error model can be used in combination with the distance-based one presented in the last section. In such a case, the detection error probability values are used to compute the original neighbor weights (w_k) to obtain new weights. These distance-based weights are then used to compute the node weight in this model.

4.5 Simulation Results

In this section, we present a set of simulation results that are intended to demonstrate some of the advantages of our fault-tolerant estimator (FTEDD). In particular, the estimator is compared to the approach proposed in [49]. The simulations were conducted using *GTSNetS*.

We simulated a sensor network of 1024 nodes randomly deployed (uniform distribution) in a region of 680 meters by 680 meters. The communication range was set to 23 meters. The parameter r defining the fault-tolerance range was set such that each interior node has 4 neighbors that are used in the distributed decision mechanism. One source (sensed object) was placed at the lower left corner of the region of interest. The sensing range was set to 93 meters. All simulation results were obtained by averaging over 1000 runs.

To reduce the size of the exchanged messages, we run an initial neighbors discovery phase prior to the execution of the fault-tolerance algorithm. This avoids having to send node locations along with every sensor reading message, which greatly reduces the sensor message size. This helps to reduce the energy cost of the algorithm, but requires the existence of an identification mechanism [69]. Every node communicates

only with nodes in its fault-tolerance range. In this specific simulation, this range is less than the communication range. Nodes, therefore, broadcast their messages to the neighboring nodes and there is no need for any routing protocol.

In this simulation scenario, we are assuming a Gaussian error term. However, our decision scheme does not require this assumption. In fact, nodes are not required to know the specific distribution of the error term prior to the deployment or to have the same distribution. The error is assumed to have a mean of 0. We simulate the case of nodes having different error probability levels by assigning to a node k a random standard deviation σ_k. This standard deviation varies uniformly in a range of plus or minus a fixed percentage of the average standard deviation σ. This fixed percentage is referred to as the variation percentage in the rest of this section. The error level corresponding to the average σ is referred to as the nominal error probability in the rest of this section. The error probability p_k is then computed using the expression in equation 21. With $m_e = 32$ and $m_n = 0$. The tail function was approximated using the erfc (error complementary) function using the following expression.

$$Q(x) = \frac{1}{2} erfc(\frac{x}{\sqrt{2}}) \tag{52}$$

The following metrics, defined in [49] are used to compare our algorithm the algorithm proposed in [49] which uses a majority voting scheme.

- Number of errors corrected: number of original sensor errors detected and corrected by the algorithm.
- Number of errors uncorrected: number of original sensor errors undetected and uncorrected by the algorithm.
- Number of errors introduced by the solution: number of new errors introduced by the algorithm.

- Reduction in errors: overall reduction in number of errors, taking into account the original errors and the ones introduced by the algorithm.

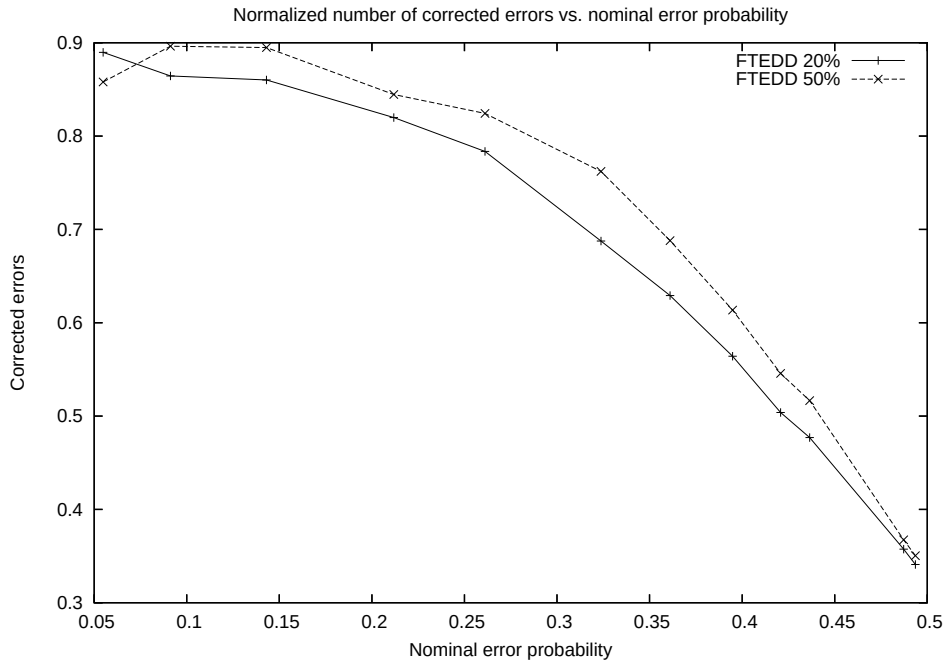

Figure 30: Normalized number of corrected errors vs. nominal error probability for 20% and 50% percentage variations

The effects of two parameters on these metrics are studied. These parameters are the nominal local detection error probability levels and the variation percentage. In Figure 30, the normalized number of original errors detected and corrected using the FTEDD estimator is plotted as a function of the nominal error rate for two different values of the variation percentage. This graph shows that the estimator corrects a large percentage of errors caused by the inaccuracy of the local node decision scheme. The estimator corrects more than 85% of the local errors for an error level as high as 15%. The plot shows also that the FTEDD estimator maintains a high level of performance as the heterogeneity of the sensor nodes, represented by the variation in standard deviation, increases. In fact, the percentage of corrected errors remains relatively unchanged when the variation goes from 20% to 50%. The slightly

better performance obtained for a percentage variation of 50% comes from the fact we vary the standard deviation σ and not the error probability itself. As the variation increases, the average error probability decreases due to the nature of erfc function. In fact, as this probability increases the variation in σ translates into less significant variation around the nominal probability of error. This is due to the decreasing rate of increase of p as a function of σ given in equation 52.

The normalized number of uncorrected errors can be readily computed from the normalized number of corrected errors, since the two sum up to 1.

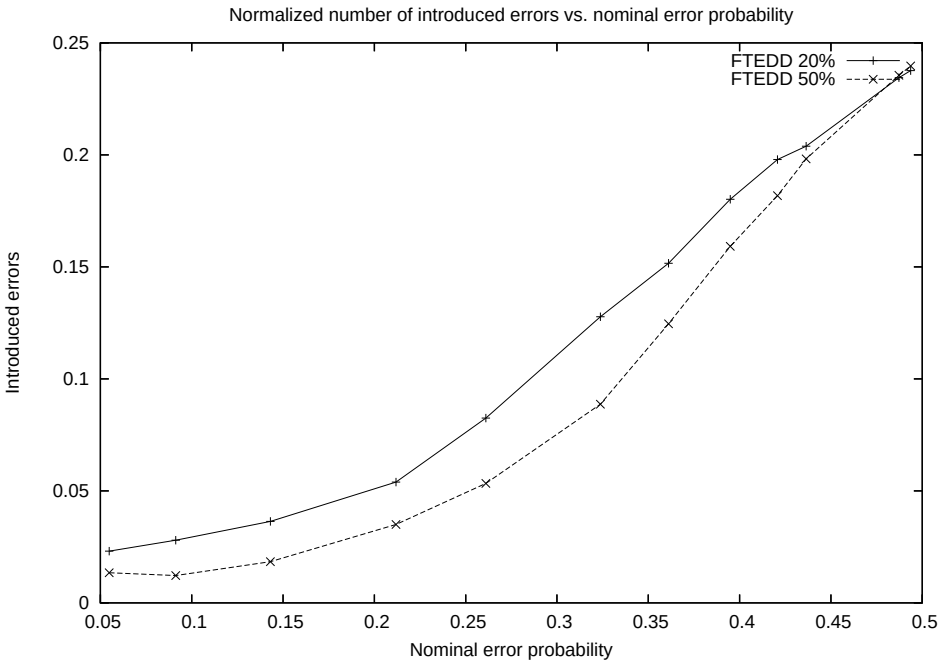

Figure 31: Normalized number of introduced errors vs. nominal error probability for 20% and 50% percentage variations

Figure 31 gives the number of errors introduced by the use of the FTEDD estimator as a function of the nominal probability. This normalized percentage is computed as the number of introduced errors divided by the number of errors present when the estimator is not used. As the figure shows, the number of introduced errors remains relatively small, e.g., less than 5% for a nominal error probability of up to

15%. Again, the performance of the estimator remains very stable with respect to the level of variation. In particular, the normalized number of introduced errors does not change much between the cases 20% and 50% variation.

The normalized reduction in errors shows that the estimator reduces greatly the level of errors. At a nominal error level of 15%, the estimator reduces the average number of decision error by more than 80% as shown in Figure 32. Again, there is a slightly better performance when the percentage variation increases from 50% to 20%.

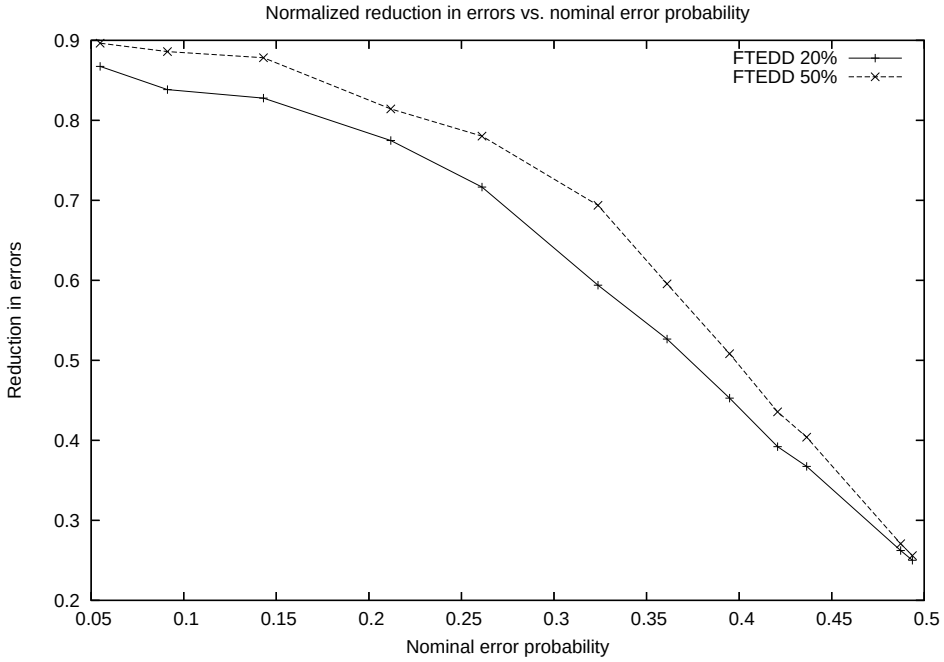

Figure 32: Normalized reduction in errors vs. nominal error probability for 20% and 50% percentage variations

Our simulations demonstrate that the FTEDD estimator performs better than the majority voting scheme of [49]. The two estimators give similar results of the normalized number of corrected errors. However, FTEDD introduces fewer new errors than does the majority voting scheme as shown in Figure 33. The difference is even greater when we increase the level of variation around the nominal probability level.

For a variation of 50%, for example, the number of errors introduced by the majority voting scheme is more than three times the number introduced by FTEDD for an error probability level of up to 15% as shown in Figure 34.

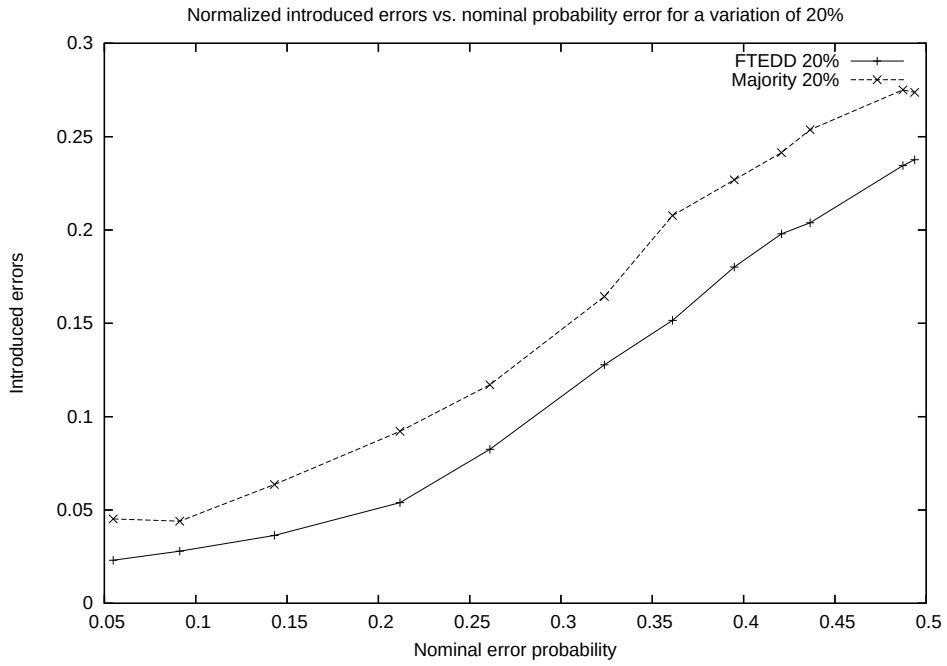

Figure 33: Normalized introduced errors vs. nominal probability error for a variation of 20%

4.6 Learning Error Probabilities

The different error models and decision schemes proposed in the previous sections require the knowledge of the error probability at each node. This probability is needed to compute the weight of each node in the decision scheme. Nodes, therefore, need a robust mechanism to learn their failure probability dynamically. This section proposes several methods to achieve this objective.

Estimation is the only realistic way to gain knowledge about the error probability at different nodes and assign appropriate weights in the distributed detection algorithm. It is clearly not realistic to assume that these probabilities are known a priori or that they remain constant through the lifetime of a sensor network. To be useful,

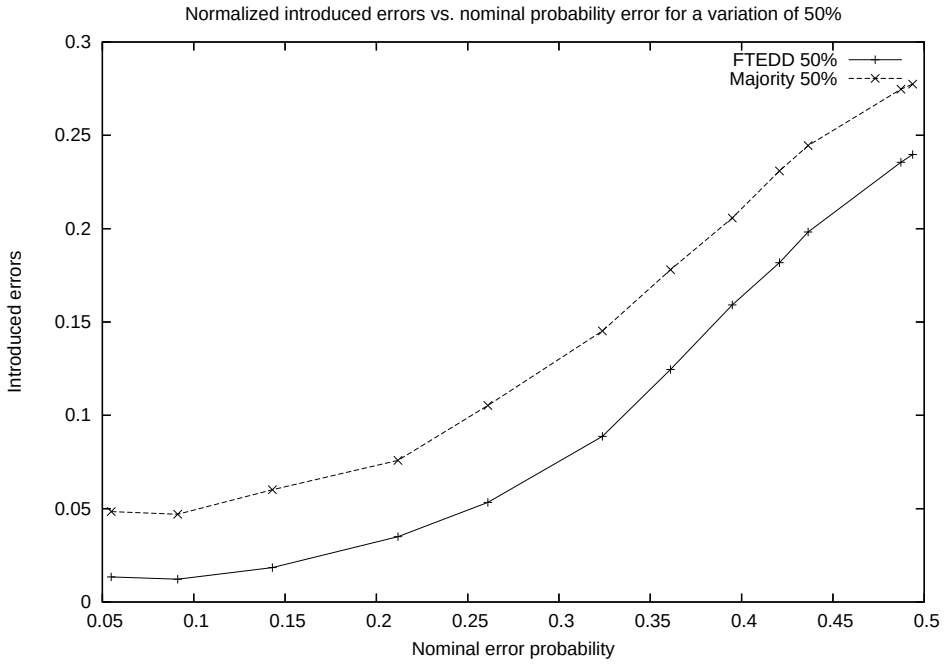

Figure 34: Normalized introduced errors vs. nominal probability error for a variation of 50%

an error probability estimation algorithm must converge to a good estimation of the true error probability in a reasonable amount of time and quickly detect real changes in error probability. Yet, the algorithm also needs to filter short-term random variations. In addition, the algorithm must not require large computing and memory resources due to the limited resources of a typical sensor node.

An error rate estimation algorithm can use the past behavior of a sensor node to estimate its error probability. This involves keeping track of how many times the node has been faulty in the last N measurements. A problem with this approach is that it requires the true situation (presence or absence of an event) to be compared with the node's decision in order to determine whether the node is faulty or not. Clearly, we cannot use the true situation (since it is not known) in the error rate estimation algorithm. Instead, we can use the weighted majority decision as an estimate of the true situation. This assumes that the initial values of node weights do

not result in a situation where nodes with high error rates are given relatively high weights (considered as more reliable) while nodes with low error rates are assigned low weights. To avoid such a situation in the first round of error rate estimation, we can start with all nodes assigned the same weights (same initial estimate of error rate for all nodes). This is equivalent to using a simple majority voting scheme in the beginning of the network deployment.

The error rate estimation scheme proposed here divides the operation time of a sensor node in sections of equal length called windows. The estimation is performed at the end of each window. Since a sensor node's local detection decision is either correct or faulty at each time step, the error rate can be found from the number of wrong decisions in the window.

Windows can be of types:

- Time window: In this case, each window contains a fixed number of detection decisions, D. Each node maintains a count of the number of detection errors, E_j, in the j^{th} time window. The error rate is estimated at the end of each time window as Er_j, which serves as an estimate of p_n of the node:

$$\hat{p}_n = Er_j = \frac{E_j}{D} \tag{53}$$

- Event window: In this case, windows are sized to contain an equal number of detection errors E, but the number of detection decisions can vary from window to window. Each node maintains a count of the number of detection decisions, D_j, until the E^{th} detection error in the current window. The error rate estimate is then given by:

$$\hat{p}_n = Er_j = \frac{E}{D_j} \tag{54}$$

The event window is particularly suitable in the case of low error rates for part of the network. In such a case, a time window approach will require setting the window

size very large to avoid resolution problems. For example, if the true error rate is 1%, then a window size of $D = 50$ can lead to time windows with alternate error rate estimates of respectively 0% and 2%.

The error rate estimates are used to replace the value of p_k for the node n and its neighbors in the FTEDD estimator, i.e., Equation 24. We now define the following functions using the estimates of the failure probability.

Definition 9 *The following two functions $\hat{F}_n^0$ and $\hat{F}_n^1$ are defined as follows:*

$$\begin{aligned} \hat{F}_n^j &: P_n \rightarrow \mathbb{R}^+, j \in \{0,1\} \\ \hat{F}_n^j(v) &= P(T_n(t) = j) \prod_{k|v(k)=j} \frac{1 - \hat{p}_{FTN_n(k)}}{\hat{p}_{FTN_n(k)}} \end{aligned} \tag{55}$$

These functions are used in the FTEDD instead of the ones defined in Definition 5. The new distributed detection error based on the new functions is given by:

$$\begin{aligned} \hat{Pe}_n(t) = P(R_i \neq T_i) = p_n(t) \sum_{v \in \hat{\Omega}_0} \prod_{k|k \in FTN_n} p_k^{1-S_k(t)}(1-p_k)^{S_k(t)} \\ +(1 - p_n(t)) \sum_{v \in \hat{\Omega}_1} \prod_{k|k \in FTN_n} p_k^{S_k(t)}(1-p_k)^{1-S_k(t)} \end{aligned} \tag{56}$$

where $\hat{\Omega}_0 = \{v \in P_n : \hat{F}_0(v) > \hat{F}_1(v)\}$ and $\hat{\Omega}_1 = \{v \in P_n : \hat{F}_1(v) > \hat{F}_0(v)\}$. The new distributed detection error probability is only different from the previous one in the way the two subsets $\hat{\Omega}_0$ and $\hat{\Omega}_1$ are computed. It must be noted that $\hat{Pe}_n(t) \geq Pe_n(t)$ since $Pe_n(t)$ corresponds to the optimal estimator as proven in Theorem 4.2.2.

The probability of the node n being in the minority quorum, i.e., disagreeing with the decision obtained using the FTEDD distributed detection scheme with the current values of the error rate estimates, is computed in the following proposition.

Proposition 3 *The probability of node n being in the minority quorum is given by:*

$$\hat{p}_n(j) = P(S_i \neq R_i | T_i, current\ estimates\ of\ p_k) = p_n + \hat{Pe}_n(t)(1 - p_n) \tag{57}$$

Proof 12 *The proof comes from the observation that the node disagrees with the majority either because the node is truly faulty and the majority is correct or the node is correct while the majority is faulty. That is:*

$$\begin{aligned} \hat{p}_n(j) &= P(S_i = T_i)P(R_i \neq T_i) + P(S_i \neq T_i)P(R_i = T_i) \\ &= (1 - p_n)\hat{P}e_n(t) + p_n(1 - \hat{P}e_n(t)) = p_n + \hat{P}e_n(t)(1 - p_n) \end{aligned} \tag{58}$$

The following proposition assesses the biasness of the window-based error rate estimator. We consider that we are using a time window. The same result can easily be proven for the event window.

Proposition 4 *The error rate estimator obtained at the end of the first window is biased. However, the estimator is asymptotically unbiased i.e., it tends to become unbiased as the number of neighbors becomes large.*

Proof 13 *For the i^{th} detection decision in the window, let's define e_i as a binary variable with value 1 if the node is seen as faulty: the node disagrees with the majority. $e_i = 0$, otherwise. The error estimation at the end of the window can be written as:*

$$\hat{p}_n^1 = Er_j = \frac{\sum_{i=1}^{D} e_i}{D} \tag{59}$$

The expected value of the p_n^1 estimate is given by:

$$\begin{aligned} E[\hat{p}_n^1] = E[\frac{\sum_{i=1}^{D} e_i}{D}] = \frac{\sum_{i=1}^{D} E[e_i]}{D} \\ E[\hat{p}_n^1] = \frac{\sum_{i=1}^{D} 1P(e_i = 1) + 0P(e_i = 0)}{D} = \frac{\sum_{i=1}^{D} P(e_i = 1)}{D} \\ E[\hat{p}_n^1] = \frac{\sum_{i=1}^{D} \hat{p}_n(j)}{D} = \frac{D\hat{p}_n(j)}{D} \end{aligned} \tag{60}$$

This results in:

$$E[\hat{p}_n^1] = p_n\hat{(}j) = p_n + \hat{P}e_n(t)(1 - p_n) \tag{61}$$

Since we assume that for all nodes $0 < p_k < \frac{1}{2}$, we have that $E[\hat{p}_n^1] > p_n$, which implies that the estimator is positively biased. However, as the number of neighbors of

n increases, the probability $\hat{P}e_n(t)$ *goes to* 0 *when using the majority voting as is the case in the beginning. For example, if the node has* 20 *neighbors, then the probability that at least* 10 *neighbors are faulty is less than* $\frac{1}{2^{10}}$*. This proves that the estimator is asymptotically unbiased.*

The next proposition gives the consistency of the window-based estimator.

Proposition 5 *The error rate estimator used to the estimate the error rate at the end of the first window is consistent, i.e., the variance of the estimator tends to* 0 *as the window size becomes large.*

Proof 14 *We have that the variance of the estimator is given by:*

$$Var[\hat{p}_n^1] = Var[\frac{\sum_{i=1}^{D} e_i}{D}] = \frac{\sum_{i=1}^{D} Var[e_i]}{D^2} \tag{62}$$

This implies that:

$$Var[\hat{p}_n^1] = Var[\frac{\sum_{i=1}^{D} e_i}{D}] = \frac{Var[e_i]}{D} \tag{63}$$

Clearly if the number of samples in the window tends to infinity, this variance will go to 0*, which proves the consistency.*

What the previous two propositions give us is that the error rate estimation obtained at the end of the first window is higher than the true error (positive bias). However, the margin can be controlled by including a larger set of neighbors. In this case, the estimator is almost centered around the real value. In addition, the consistency proposition guarantees that if the window size is large enough then this estimate does not vary much around its mean. This means that the estimator that we obtain using one sample population (window) is close to the mean estimate and therefore to the real value.

The previous two propositions can be used to obtain the following corollary on the convergence and stability of the window-based error rate estimator.

Corollary 3 *The window-based estimator converges and is stable when the number of neighbors is large and the window size is large.*

Proof 15 *The proof comes from the fact that the estimator is asymptotically unbiased and consistent at the end of the first window. Which implies that it converges as for a large enough neighborhood (asymptotic unbiasedness) and large enough window (consistency). The stability comes from the fact that once the convergence is achieved, the new FTEDD estimator (optimal with respect to the MAP criterion) has a lower distributed detection error probability than the simple majority used in the first window. This in turn implies asymptotic unbiasedness and consistency in the second time window and so on.*

This theoretical analysis of the biasness and consistency of the estimator and the importance of the window size for the convergence and stability of error rate estimation is confirmed through simulation. Figures 35 and 36 give the performance of the error rate learning as a function of the window size. The simulation scenario consists of a sensor network of 1024 nodes and a density of 4 neighbors per node as in Section 4.5. The nominal error probability is 0.143 corresponding to a standard deviation of $\sigma = 15$ with 60% error probability variance. The figures show the number of nodes that converge at the end of the first window. A node is considered to have converged if the error rate estimate is close to the true error rate (within ± 0.025).

As expected, we can see that the convergence rate increases with the window size. As shown in the figures, accurate and stable error rate estimation requires large window sizes. It must be noted here that in both figures, few nodes (3 or 4 our of the 1024 nodes) do not converge. These nodes are located in the border of the event region with neighbors on both sides report opposite detection decisions without being faulty. As discussed earlier, this is not a serious problem as long as it affects only a marginal number of nodes.

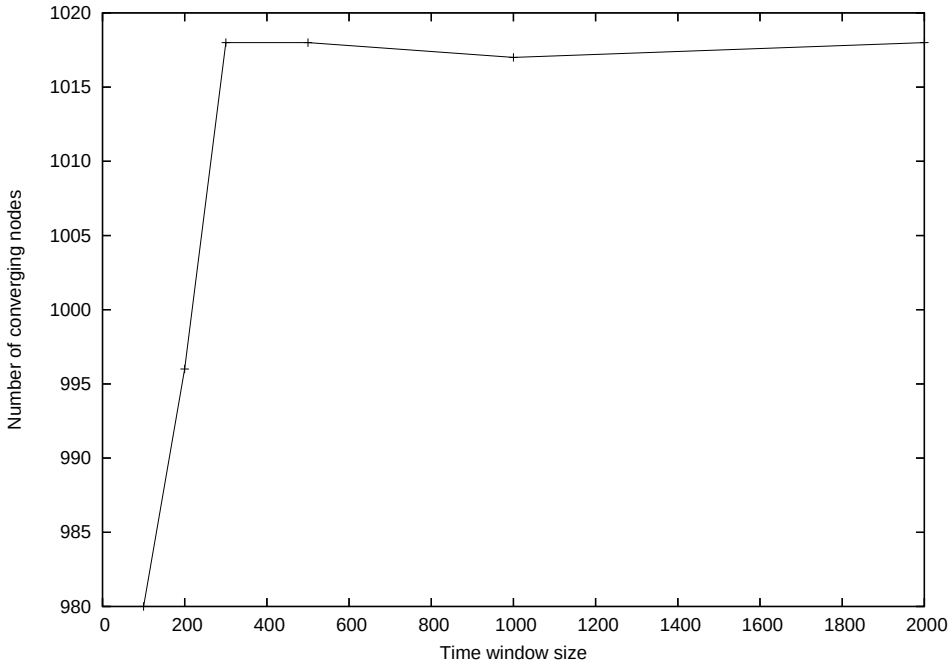

Figure 35: Time window error estimation convergence

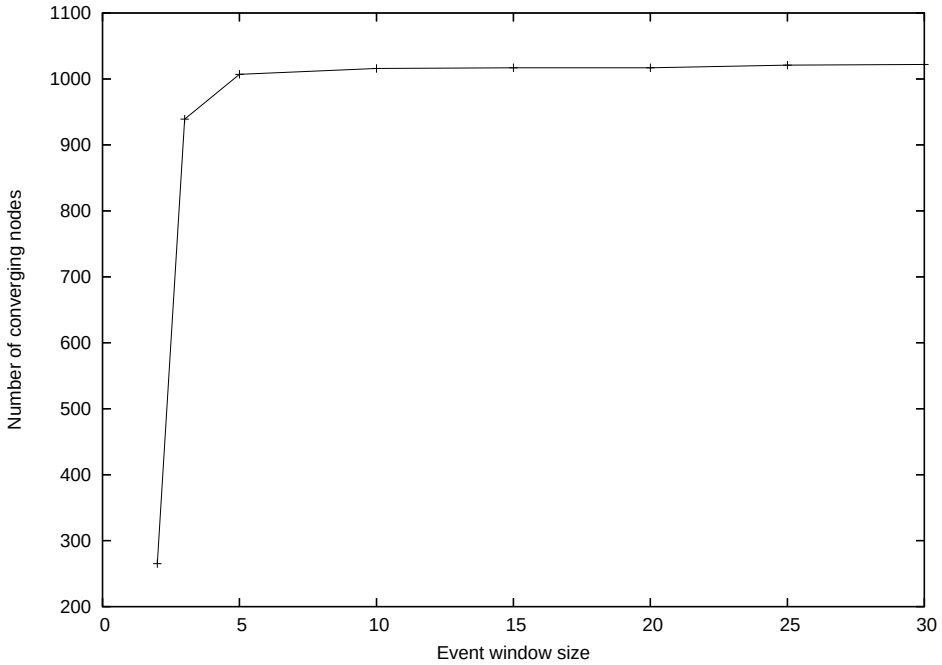

Figure 36: Event window error estimation convergence

Despite resulting in a more accurate error rate estimation, large window sizes mean that the error estimation is not performed as often, which means that the estimation algorithm cannot react quickly to a change in the error rate. To handle this problem, we propose to use smaller window sizes to update the error rates more often then filter the rates from several windows to smooth the results. We are virtually increasing the window size by including more windows in the error rate estimation, which is necessary for consistency. This can be achieved by using the Er_j estimate from the m previous windows to compute a more accurate error rate estimate pE_j at the end of the j^{th} window. Two approaches can be used: the regular moving average and the geometric moving average.

4.6.1 Moving Average

In this case, each node averages the window error rate estimates from the previous m windows to compute pE_j by averaging over these estimates:

$$pE_j = \frac{1}{m} \sum_{l=j-m+1}^{j} Er_l \tag{64}$$

The moving average estimator is a low-pass filter. For example, Figure 37 gives the frequency response of the low-pass filter corresponding to the moving average filter when $m = 10$. This is sometimes referred to, in literature, as a 10-point moving average filter.

As expected, the performance of the moving average estimation scheme is highly dependent on the value of the parameter m. This parameter must be set to balance between two objectives. The estimation scheme must have good smoothing action to reduce the effects of random noise being considered as a real change in error rate (stability). At the same, the moving average filter needs to have a large enough bandwidth to detect real changes in error rate such as increase in number of errors because of aging or a sudden increase in number of detection faults. The bandwidth

must be determined based on a frequency profile of error rates. As the value of the parameter m increases, the estimation scheme becomes more robust to random noise but at the same time the filter bandwidth decreases. This can be seen from Figures 37, 38 and 39 displaying the frequency response of the corresponding low-pass filters for $m = 10$, $m = 20$ and $m = 40$.

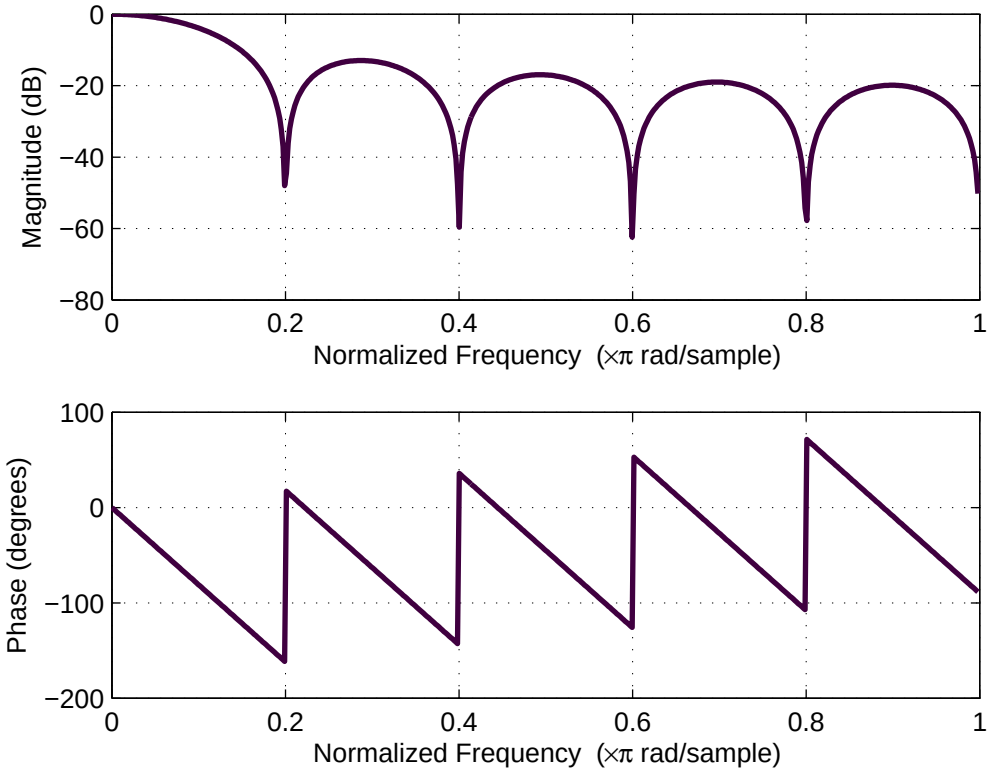

Figure 37: Frequency response of the moving average filter for m = 10

It must be noted that the moving average can be implemented in an efficient way not requiring extensive computing resources. In fact, a simple recursive implementation involving only 2 additions instead of m additions is readily available. This involves replacing the estimation equation by:

$$pE_j = pE_{j-1} + \frac{Er_j - Er_{j-m+1}}{m} \tag{65}$$

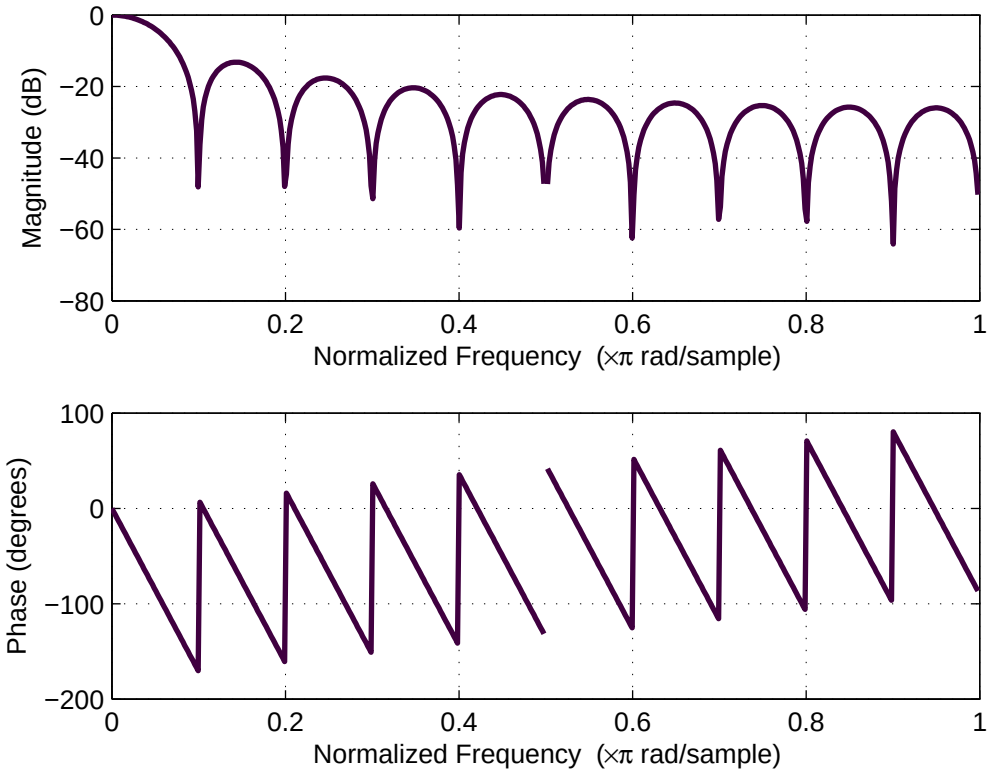

Figure 38: Frequency response of the moving average filter for m = 20

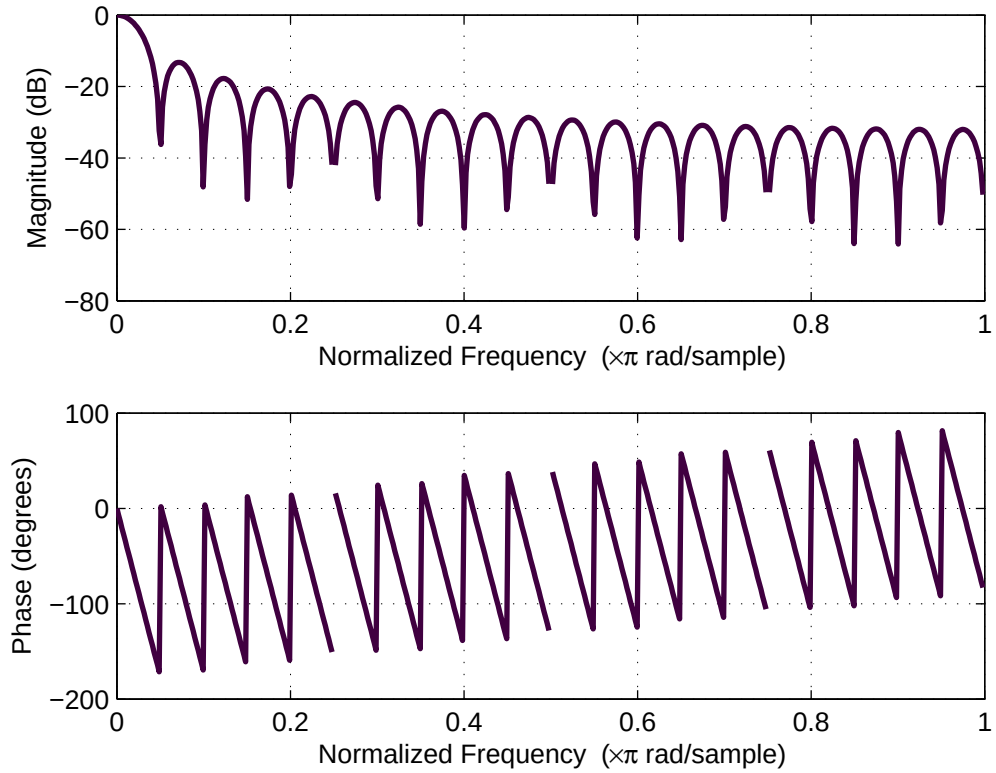

Figure 39: Frequency response of the moving average filter for m = 40

4.6.2 Geometric moving average

This technique also called the exponential moving average (EMA) performs a recursive geometrically weighted error rate estimation:

$$pE_j = bEr_j + (1-b)pE_{j-1} \tag{66}$$

where $0 \leq b \leq 1$. This is equivalent to a geometric distribution with parameter b since pE_j can also be formulated as:

$$pE_j = b\sum_{l=1}^{l}(1-b)^{j-l}Er_l + (1-b)pE_0 \tag{67}$$

In the case of the geometric moving average, the relationship can still be interpreted approximately in similar way to the moving average with a sliding number of m windows. The parameter b can be set so that an arbitrarily large percentage of the probability mass lies in the previous m windows. For example, to have 99% of the probability mass function lie in the previous m windows, the parameter b must be set according to the following relationship:

$$b\sum_{l=1}^{m}(1-b)^l = 1-(1-b)^m \simeq 0.99 \tag{68}$$

This results in:

$$b = 1-(0.01)^{\frac{1}{m}} \tag{69}$$

In the same way as the simple moving average, the geometric moving average can be represented as a low-pass filter. In this case, the value of the b parameter is key to the performance in terms of how quickly changes in error rates are detected and accounted for as well as in terms of the robustness of estimation scheme to random changes. Figures 40, 41 and 42 give the frequency response of the low-pass filter corresponding to a geometric moving average for $b = 0.10$, $b = 0.25$ and $b = 0.37$, respectively.

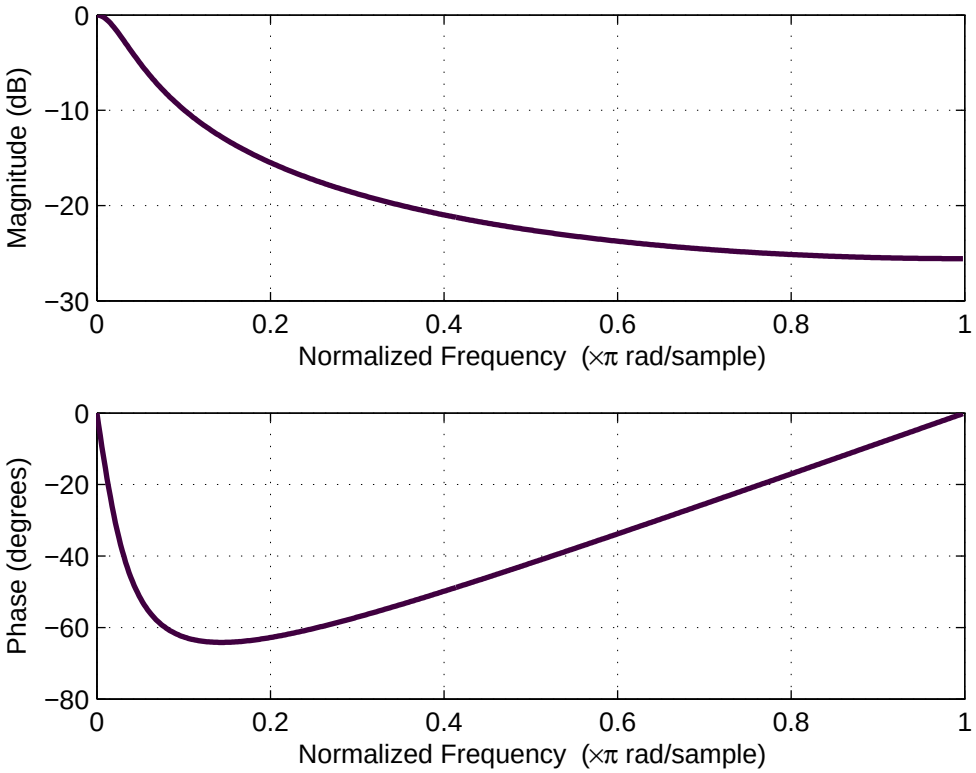

Figure 40: Frequency response of the geometric moving average filter for b = 0.10

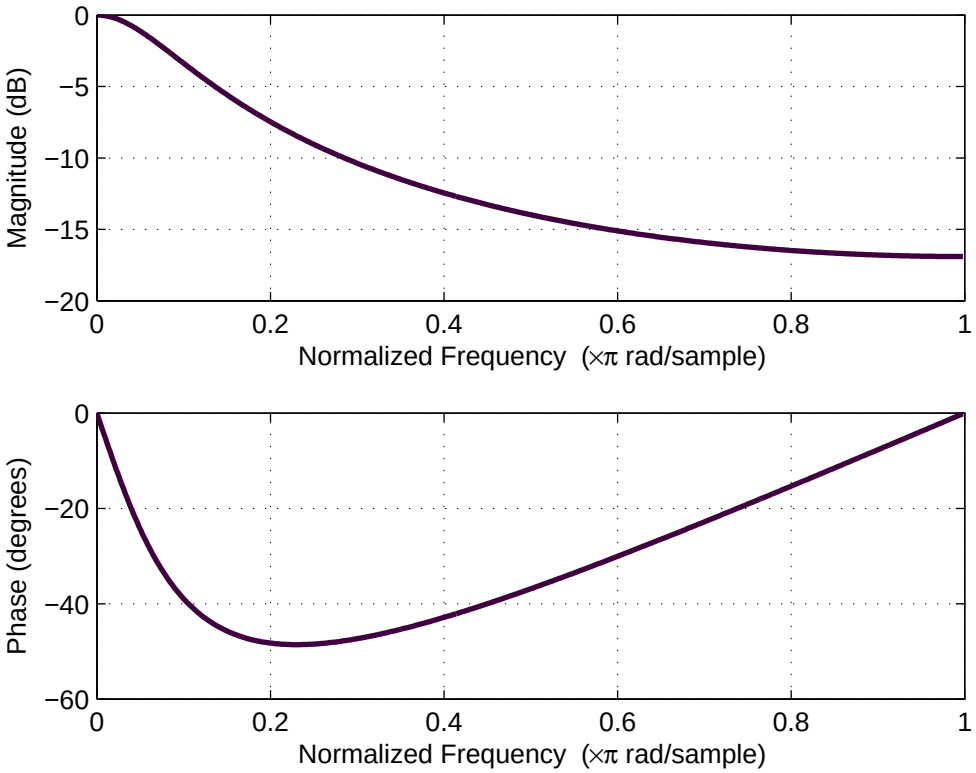

Figure 41: Frequency response of the geometric moving average filter for b = 0.25

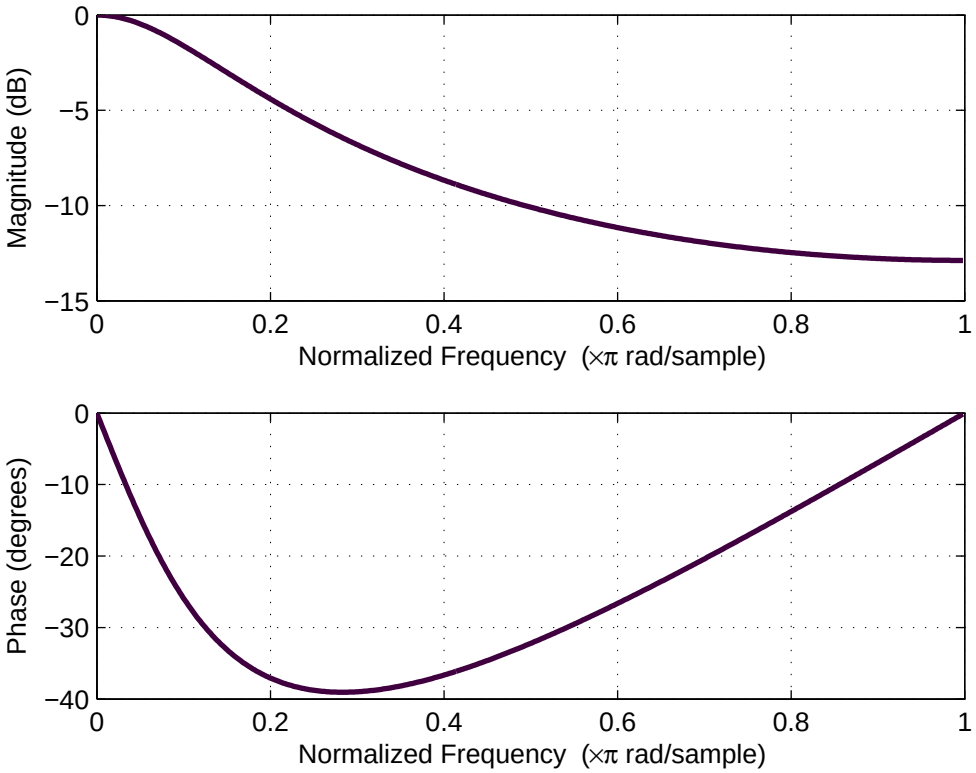

Figure 42: Frequency response of the geometric moving average filter for b = 0.37

The previous figures show that the geometric moving average has a small phase lag indicating smaller time lags compared to the simple moving average filter (Figures 37, 38 and 39). When using an equivalent number of windows, the choice between the moving average and the geometric moving average represents a tradeoff between the faster reaction to changes in the error rate that can be achieved with the geometric moving average and the higher robustness to random changes that is more likely when using a simple moving average scheme. This is because the moving average gives equal weights to all the previous m windows while the geometric moving average gives higher weights to closer windows.

4.6.3 Simulation Results

For accuracy and fast response of the estimates of the error rates, it is clear that the choice of the values of the parameters b and m as well as the choice of the initial value pE_0 are important. As seen from the frequency response of corresponding filters, if b and m are chosen so that few recent windows are dominating the estimate, the risk is that the node takes random noise as real change in error rate. This can result in increased number of distributed detection errors since the corresponding is given the wrong weighting in the weighted majority voting scheme. In contrast, if too many past windows are taken into account in the estimation of the error rate, real change in error rate will not be detected in a timely manner. The initial value of the error rate estimate affects how quickly the error rate estimation scheme converges after the network deployment. This subsection presents simulation data to assess the effects of these different parameters on how quickly the learning schemes converge and how robust it is to random changes.

In all the simulation scenarios, we use the same sensor network as described in Section 4.5. The time window size is fixed at 100 samples per window.

We first evaluate the effect of the parameter m on the performance of the simple

moving average error rate learning scheme. Figure 43 gives the convergence rate as a percentage of nodes convergence rate as a function of the number of windows taken into account in the moving average. True values of the error rates are set with a nominal error probability of 0.143, which corresponds to a standard deviation of $\sigma = 15$. The error rate variance is 60%. The window size is set to 100 samples (detection decisions). Results are given as number of nodes that converged within 10, between 10 and 20 windows and after 20 windows. As can be seen, the smaller the value of m the faster is the convergence. For example, the number of nodes converging in the first 10 windows decreases as m increases. On the other hand, the number of nodes of nodes converging after 20 windows increases with the value of m. This is not surprising since a smaller number of windows taken into account in the estimation means that the dependence on the initial value of the error rate estimate is removed faster.

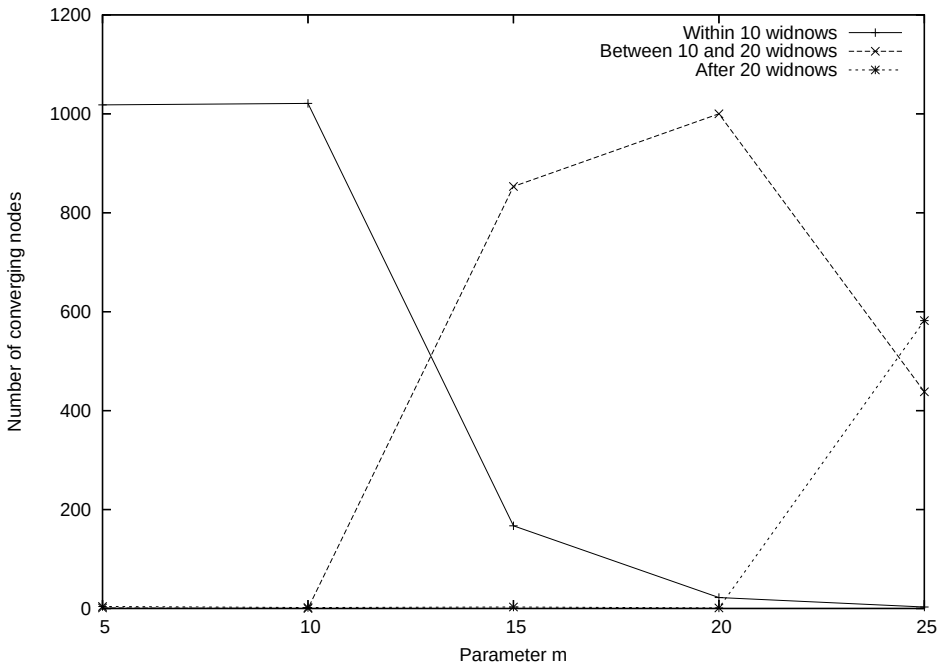

Figure 43: Rate of convergence (percentage of nodes that converge to true error rate) of simple moving average vs. m

Figure 44 plots Robustness metric, the percentage of nodes that remain stable near the true error rate after experiencing convergence during the first 20 windows as a function of the parameter m. Clearly, when the moving average scheme uses a small number of windows the learning algorithm is more vulnerable to random changes. This means that because the weight given to the current window ($\frac{1}{m}$), a random variation in number of errors during this window can be misinterpreted as a valid change in error rate.

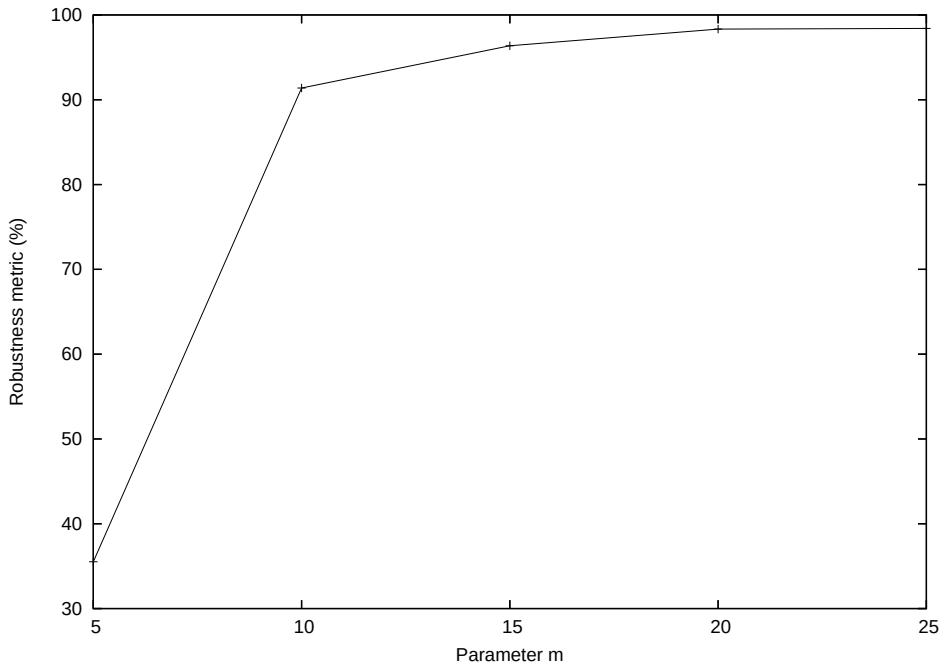

Figure 44: Robustness of the the simple moving average to random changes, where Robustness metric is the percentage of nodes that stay close to true error rate after convergence

Clearly, the parameter m should be set by taking into account any prior knowledge of the network deployment characteristics. If it is expected that the initial estimate of the error rate is reasonably accurate and that the error rate will evolve very slowly, then a large value of m is more reasonable. If on the other hand, there is little information available to initialize the error rate estimate accurately and a fast convergence is important, then the parameter m should be set to a small number of windows. An

alternative could be to set the parameter initially to a small value to allow fast convergence in the beginning of the algorithm and then change it to a larger value since it is likely that any change in error rate will occur over a long period of time.

In the case of the geometric moving average, the most important parameter is the weight b of the current window. Figure 45 gives the convergence rate of the geometric moving average learning scheme for different values of the parameter b. The percentages of nodes converging during the first 10 time windows, between 10 and 20 windows and after more than 20 windows are given. As expected, the convergence happens faster for higher values of b.

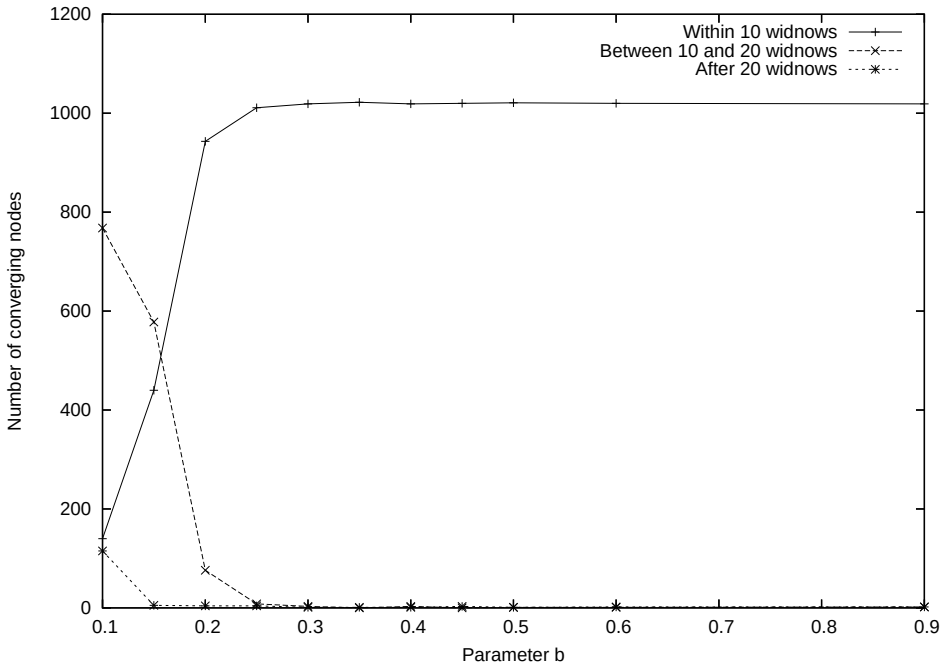

Figure 45: Rate of convergence (percentage of nodes that converge to true error rate) of geometric moving average vs. b

Figure 46 presents the results for the robustness of the geometric moving average to random fluctuations in error rate for various values of the weight parameter b. The figure plots *Robustnessmetric*, the percentage of nodes that deviate by 0.025 or more from the true error rate after converging within the first 20 time windows. As

previously seen in the frequency domain representation of the corresponding low-pass filter, high values of b cause the error rate estimation to be sensitive to random noise around the true error rate.

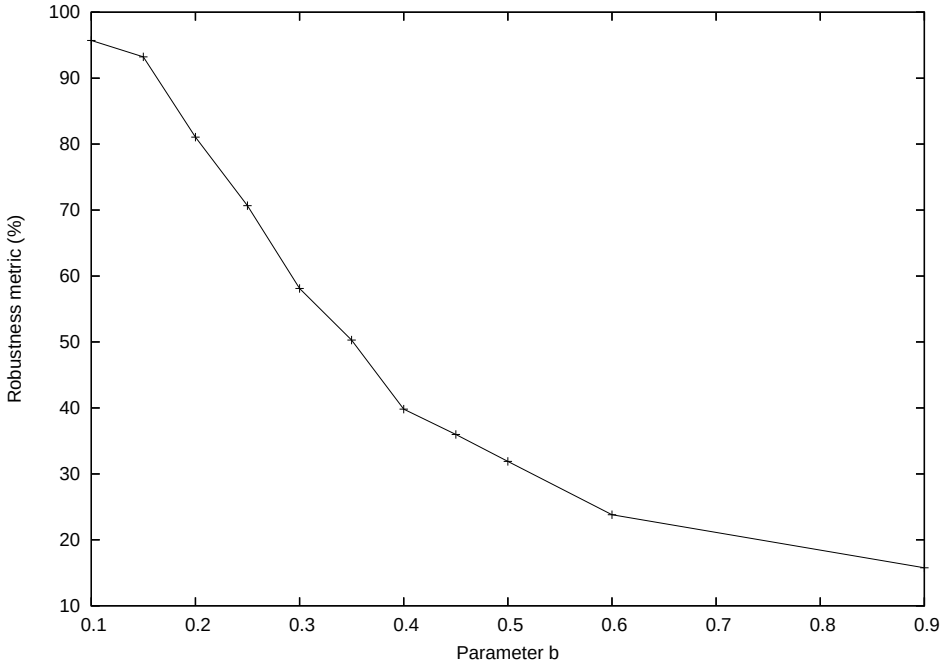

Figure 46: Robustness of the the geometric moving average to random changes, where Robustness metric is the percentage of nodes that stay close to true error rate after convergence

As for the parameter m, the weight factor b must be set to balance between fast convergence and robustness against random fluctuations around the true error rate.

We next look at the influence of the initial estimate on the convergence and robustness to changes. We use a geometric moving average scheme with the parameter b set to 0.37. This corresponds to the case where at least 99% of the probability mass function of the corresponding distribution lies within 10 nearest time windows. Figure 47 gives the convergence rates within 10 windows, between 10 and 20 windows and after 20 time windows for different values of the initial rate estimate. The convergence is faster when the initial value of the error rate estimate is between 0.10 and 0.15. This is not surprising since the true error rate of the nodes in the network is

chosen randomly from this interval. It must be observed that this parameter does not have as much influence as the parameter b. Even though it affects how affects how fast the convergence is reached, this effect is marginal compared to that of the parameter b. The initial value of the error rate estimate does not show to have significant effect on the robustness to error change.

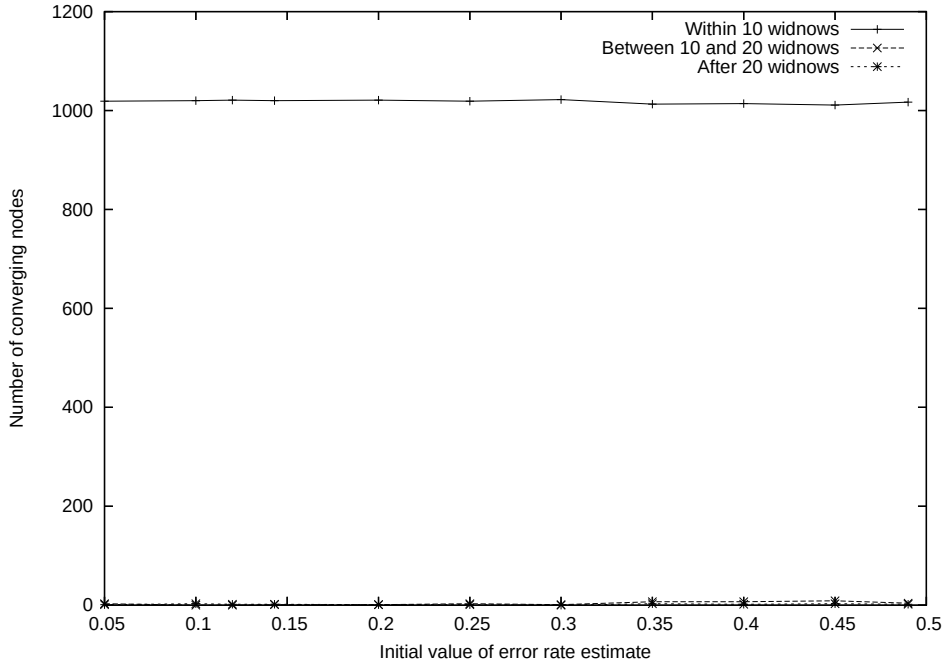

Figure 47: Convergence of geometric moving average vs. initial error rate estimate

In summary of this section, we have proved that the error rates of individual nodes can be learned in a dynamic way. This is necessary for the performance of the FTEDD estimator since it is based on knowledge of node weights in the voting scheme. Node weight is directly derived from the error rate. We also proved the importance of setting the learning scheme parameters appropriately for fast response time and for robustness to random rate changes.

4.7 Conclusion

We presented an optimal fault-tolerant estimator for distributed detection in sensor networks with sensor nodes of different accuracy levels. This estimator is proven to be equivalent to a weighted voting scheme. We also provided two new error models that account for the node distance and the geographical quorum distribution in the distributed detection decision scheme.

In addition to the theoretical analysis, the proposed fault-tolerance event detection scheme was tested and gave good performance under various simulation settings. It was found, for example, that this scheme can detect and correct more than 85% of original detection errors, while introducing only less than 5% of new errors.

Finally, we also proposed schemes based on the moving average and the geometric moving average to allow nodes to efficiently learn and update their error rates. These rates are necessary to compute node weights used in the weighted voting scheme.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

This work described three main contributions to the growing field of sensor network algorithms. The first contribution consists of a simulation tool (the *Georgia Tech Sensor Network Simulator*, *GTSNetS*) that enables the design and evaluation of new algorithms for sensor networks. The second contribution of this work consists of a novel approach to the global ID assignment problem. The third contribution is a fault-tolerance detection mechanism that is proven to be optimal under the MAP (maximum a posteriori) criterion.

5.1 Sensor Network Simulation

Chapter 2 described the design and implementation of *GTSNetS* (*Georgia Tech Sensor Network Simulator*) and illustrated how it can be used to study sensor networks. The major contribution here is a new simulation environment that allows efficient study and comparison of sensor network designs and architectures. This simulator is designed to closely match how typical sensor nodes and networks function. It models sensor nodes consisting of three main functional units: computing unit, communication unit and a sensing unit composed of one or several sensors. Each node is powered by a battery. Several energy consumption models are implemented for each one of the functional units.

GTSNetS is built on top of the *Georgia Tech Network Simulator* (*GTNetS*), a full featured general-purpose wired and wireless networks simulation environment. *GTNetS* is designed with the objective of closely mimicking the way real networks are structured while handling the simulation of large networks. For example, there is a clear separation between the protocol stack layers as in standard network protocols.

GTNetS has models for the various components of a network including nodes, network interfaces protocol layers and many popular network protocols. It also provides models for end-to-end user applications. Packets are modeled as composed of a list of protocol data units (PDUs). PDUs are appended and removed from the packet as it moves down and up the protocol stack.

GTSNetS extends *GTNetS* to enable sensor network simulation. Examples of new models include:

1. Models for each of the different functional units composing a sensor node: sensing unit, computing unit and communication unit.

2. Sensing accuracy models to account for sensor measurement errors. These models are useful for the simulation of sensor network applications where reliability and fault-tolerance are important metrics.

3. Battery model to track the evolution of the energy resources available to a sensor node.

4. Several energy models for each functional unit to quantify the energy cost of each task executed by a sensor node. These models are also useful to quantify the effects of different protocols and algorithms on sensor node and network lifetime.

5. Sensor network specific protocols. Several sensor network routing protocols such directed diffusion and different geographical routing protocols are modeled. New models can be easily added.

6. New tracing capabilities to enable the collection of statistics on energy consumption at network, node and functional unit levels. This capability is particularly useful for finding sources of lifetime optimization and quantify their impact. For example, it can be used to find regions of the network where nodes die quickly

because of high communication energy cost during the execution of a specific algorithm.

7. Actuator and plant models to enable the simulation of networked control systems where a sensor network is used in the context of a distributed control application.

One of the main features of *GTSNetS* is that it builds on the design of *GTNetS* for an excellent scalability to network size. In fact, it has been demonstrated, through simulation results, that it can be used to simulate sensor networks of several hundred thousand nodes. Scalability is achieved through careful resource optimization (e.g., reduction of the number of outstanding events, memory management and optimization and reduction of log files size). In particular, *GTSNetS* is implemented in an efficient and modular manner leveraging the specifics of sensor networks for higher scalability. For example, in sensor networks communication is performed via wireless devices and the network topology can be dynamic with packets transmitted typically in a localized broadcast. By observing that nodes do not need to maintain and update routing tables for most sensor network protocols, *GTSNetS* reduces the memory and processing requirements. In addition, *GTSNetS* extends *GTNetS* approach of fine-grained control of the logging of simulation events. By implementing many tracing options (such energy states, message exchange, node wake-up times, etc) and allowing the user to choose exactly which ones to log (depending on the simulation objective) and discard the others, *GTSNetS* allows for a trade-off between memory requirements and the amount of simulation events logging.

Other important features of *GTSNetS* include the fact that it is modular and easily extensible by users to implement additional architectural designs and models. This simulator allows the user to choose among different implemented alternatives including different network protocols, different types of applications, different sensors,

different energy and accuracy models. Such modularity allows an independence between the different modules, which makes it easy for the user to extend the simulator by adding new models (inherited from the existing ones) or modifying existing models without affecting other parts of simulator. The modularity and ease of use make *GTSNetS* suitable for simulating sensor networks since such networks are application-dependent and their diversity cannot be represented in a single model.

GTSNetS is distributed under the GNU General Public License and can be obtained from the web page at `http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/download.htm`.

5.2 Global ID Assignment

We presented a solution to the global ID assignment problem in sensor networks in Chapter 3. The major contribution is an algorithm that assigns unique IDs to each node participating in the network using the minimum number of bytes required to code these IDs. This solution was obtained using a three-phase approach. In the first phase, temporary long IDs are assigned. These temporary IDs are used in the second phase to determine reliably the exact size of the network and, therefore, the minimum number of bytes to use. In the third phase, final IDs coded using the minimum number of bytes are assigned.

The algorithm is further extended to allow nodes to join the network asynchronously and to obtain unique IDs after the initial deployment. This is performed at the local level by having each node initially participating in the algorithm request several spare IDs to account for neighbors that join after the initial deployment.

The termination and correctness properties of the proposed algorithm as well as its communication energy cost were studied and discussed analytically. We proved that the algorithm terminates if nodes that participate in the initial round of the algorithm remain active until after the end of the third phase of the algorithm. The

algorithm does not terminate if participating nodes die or go into a sleep state during its execution. It is very unlikely that nodes will run out of energy during the execution of the initial round of the algorithm since this execution is assumed to occur early after the network deployment when nodes still have high energy levels. In addition, we analytically proved that the probability of message retransmission is very low during the execution of the algorithm, which bounds the energy cost of the algorithm making it even more unlikely that nodes will die during its execution. The correctness of the algorithm was determined by proving that the probability of two nodes being assigned the same ID is extremely low. The occurrence of such a situation was found to depend on the joint occurrence of a set of independent events, each having a very low probability of occurrence.

To confirm the analytical results, extensive simulation results were also provided to study the effects of various network characteristics (size, density and bit error rate) on the performance of the algorithm. The effects of the time wait parameter on the the algorithm performance was also studied. This parameter is used in a collision handling mechanism implemented to minimize message losses and retransmission, which limits the communication energy consumption of the algorithm and reduces its execution time.

The theoretical and simulation analysis demonstrated that the proposed algorithm achieves excellent results, both in terms of the percentage of participating nodes receiving a unique ID and the execution time as well as in terms of communication energy consumption. These results are obtained when the algorithm parameters, in particular the time wait parameter, are set based on the network characteristics such as size and density.

5.3 Fault-Tolerant Event Detection

In Chapter 4, the major contribution is a localized algorithm that provides an optimal fault-tolerant estimator for distributed event detection in sensor networks with sensor nodes of different accuracy levels. The proposed solution leverages the spatial correlation between the occurrences of the detected event in the sensing ranges of neighboring nodes to formulate a fault-tolerance detection algorithm at the node level. In particular, instead of only using its local sensor measurement to decide on the presence or absence of an event, a node takes into account the local detection decisions of its neighbors. The node divides its neighbors into two groups depending on their local decision. The node adds itself to the group of neighbors that agree with its local decision (presence or absence of an event based on node's own sensor measurement). A confidence metric is computed for each group taking into account its members detection error probabilities. The node decides on the presence or absence of the detected event based on which group shows a higher confidence level.

Unlike previous approaches, the new approach does not assume that all the nodes have the same detection probability of errors or that this probability is known in advance. It is designed to handle the case of neighboring nodes with different reliability levels (different detection error probabilities). Relaxing the assumption that nodes have the same reliability level also allows the handling of cases where the sensor network is composed of heterogeneous sensor nodes.

We proved analytically that the detection estimator based on the new algorithm is optimal with respect to the maximum a posteriori (MAP) criterion. The estimator is also proved to be asymptotically unbiased. In contrast with existing fault-tolerant detection schemes where a simple majority voting applies, the proposed solution has been found to be equivalent to a weighted voting scheme with each neighbor given a weight that is a function of its reliability level.

Taking into account different reliability levels for each node results in the need

for an approach to estimate each node's reliability level since it cannot be assumed to be known in advance. A node's reliability level can also change over time due, for example, to sensor aging. We discussed schemes based on the moving average and geometric moving average to allow each node to efficiently learn its error rate. When the sensor network is first deployed, these proposed schemes can be used by nodes to quickly learn their error rates and establish the weight of each node in the weighted voting event detection algorithm. Using these schemes, nodes can also dynamically discover changes in reliability levels and adjust the weights properly.

In addition to the theoretical analysis, the proposed fault-tolerant event detection scheme was tested using simulation and gave good performance under various simulation settings. It was found, for example, that this scheme can detect and correct more than 85% of original detection errors, while introducing only less than 5% of new errors.

Further, we describe two new error models that take into account the neighbor distance and the geographical distributions of the two decision quorums. These models are particularly suitable for detection applications where the event under consideration is highly localized.

5.4 Future Work

5.4.1 Sensor Network Simulation

An important area of future work is the inclusion of hybrid simulation capabilities. Hybrid simulation will allow interaction between real sensor nodes and nodes in the simulator. Real nodes will be a source for real data for the simulator. This will also allow a better understanding of the effect of new architectural choices (e.g., network protocols, collaboration strategies for example) by providing the benefit of real world implementation on one or few nodes without sacrificing the ability to test on large networks modeled in the simulator. Another important future work is to test the

realism of the models implemented in *GTSNetS* by comparing their results to the outcome of real experiments on sensor networks. The hybrid simulation can enable this validation since it allows the comparison between the behavior of a real world sensor node and a node in the simulator.

5.4.2 Global ID Assignment

The ID assignment algorithm is part of the overall sensor network initialization and infrastructure building problems. In the future, it will be interesting to investigate if various initialization tasks can be performed in parallel with the ID assignment algorithm. For instance, it could be possible to establish clusters between neighboring nodes using the temporary unique IDs. This can be done after the local completion of the first phase of the algorithm. It allows the reduction of the overall cost of network initialization since the nodes will be performing initialization tasks concurrently. A potential downside of this approach is that it can increase concurrent traffic in the network, which can result in higher message retransmission rate. It will be interesting to investigate the tradeoff between the reduction in initialization time and the added communication cost.

5.4.3 Fault-Tolerant Event Detection

In the future, it would be interesting to investigate a geometric approach to fault-tolerant distributed detection in sensor networks. The idea is to track topological inconsistencies and use them to determine the presence or not of a local detection error. For example, if we assume that node locations are known and that the event region is convex and continuous, then the following observations can be made:

1. If all neighbors of a node n are detecting an event and n is not detecting this event, then there must be a detection error at n or among its neighbors. For example in Figure 48, when all nodes except n detect the event, this could indicate that the node n is faulty. This observation comes from the fact that

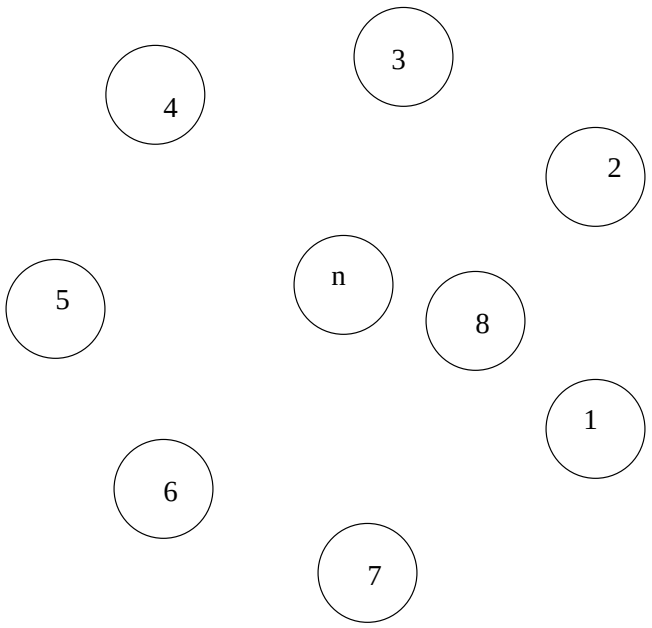

Figure 48: All nodes except n or 8 detect the event

because of the convexity assumption the event cannot be present at all neighbors of n and not at n itself. This observation can be used to allow nodes to detect their detection faults with a high probability.

2. If an event is detected by a node n and all of its neighbors except one interior neighbor, there must be a local detection error at one or several neighbors of n. This situation occurs, for example, if n and all its neighbors except node 8 in Figure 48 detect the event. This observation is again due to the assumptions of convexity and continuity of the event region. It can be used by the node n to detect errors at its neighbors.

3. It can be proven that if 3 nodes $n1$, $n2$ and $n3$, as in Figure 49 detect the presence of an event, then any node in the interior triangle ($X1X2X3$ must detect this event. In a similar way, if these three nodes do not detect an event, a node in the interior triangle should not detect it. The interior triangle is formed as follows. We define the point $Oj, j \in \{1, 2, 3\}$ as the location of node

j. The circle C_j is the circle centered at Oj with a radius equal to the sensing range. We define the point $X_{ij}, i, j \in \{1, 2, 3\}$ as the intersection the circle C_i and the line $OiOj$. There two such points on each circle as in the Figure 49. The point $Xj, j \in \{1, 2, 3\}$ is defined as the intersection of the two tangents of the circle C_j at the two points $X_{ji}, i \in \{1, 2, 3\} \setminus j$. The interior triangle is the triangle defined by the points $X1X2X3$. It can be proven that if the three nodes $nj, j \in \{1, 2, 3\}$ have the same local detection decision, then any node inside this triangle must have this same decision when no error is present. It is of importance that this observation does not require that the three trusted nodes be within a minimum distance of each other in the case of a continuous event-region. This observation can be used to place a number of "trusted" nodes in strategic locations in the deployment region so that every node in the network can detect when it is faulty. We should note here that if a regular (non-trusted) node is surrounded by three trusted nodes, it can be viewed as a trusted node after verifying its decision by comparing it with the ones of the three trusted neighbors.

These observations can be used as a starting point to develop a more rigorous geometric approach to fault-tolerance in distributed detection applications of sensor networks. Such an approach can potentially provide a deterministic approach to fault-tolerance at the expense of the assumption of node location awareness. It will also be interesting to investigate the implications of relaxing the convexity assumption.

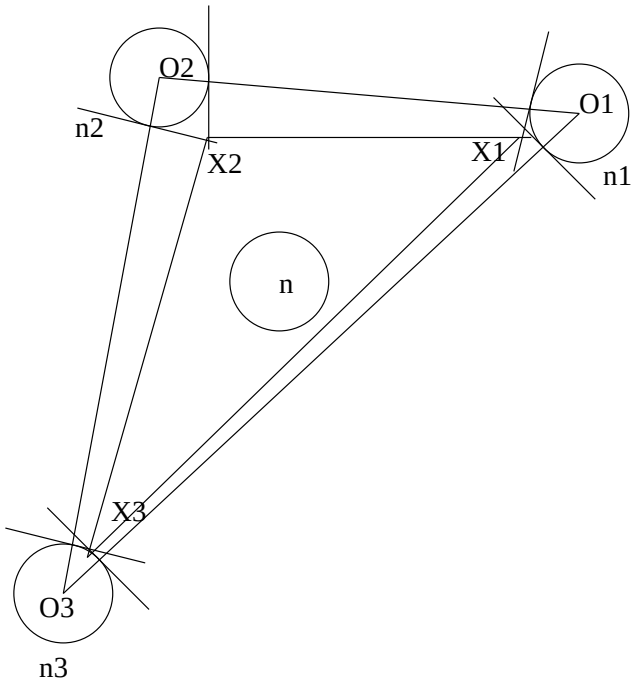

Figure 49: Node n in the interior triangle of three trusted nodes

REFERENCES

[1] "The aware home project." Website: http://www.awarehome.gatech.edu/, Accessed in March 2007.

[2] "The intelligent home project." Website: http://dis.cs.umass.edu/research/ihome/, Accessed in March 2007.

[3] "The smart kindergarten project." Website: http://nesl.ee.ucla.edu/projects/smartkg/, Accessed in March 2007.

[4] Abowd, G., Bobick, A., Essa, I., Mynatt, E., and Rogers, W., "The aware home: Developing technologies for successful aging," in *In Proceedings of AAAI Workshop on Automation as a Care Giver*, 2002.

[5] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E., "Wireless sensor networks: a survey," *Computer Networks: The International Journal of Computer and Telecommunications Networking, Volume 38 , Issue 4*, 2002.

[6] Ali, M. and Uzmi, Z. A., "An energy efficient node address naming scheme for wireless sensor networks," in *INCC*, 2004.

[7] Althaus, F. and Zhou, H., "Energieeffiziente, drahtlose sensornetze-auswirkung der hardware auf das protokolldesign," in *Verteilte Systeme-Sensornetz '2003*, IEEE Computer Society, 2003.

[8] Ammer, J. and Rabaey, J., "Low power synchronization for wireless sensor network modems," in *IEEE Wireless Communication and Networking Conference*, 2005.

[9] Bapat, S., Kulathumani, V., and Arora, A., "Analyzing the yield of exscal, a large-scale wireless sensor network experiment," in *Proceedings of the 13TH IEEE International Conference on Network Protocols (ICNP'05)*, 2005.

[10] Barriac, G., Mudumbai, R., and Madhow, U., "Distributed beamforming for information transfer in sensor networks," in *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN'04)*, 2004.

[11] Bhardwaj, M. and Chandrakasan, A. P., "Bounding the lifetime of sensor networks via optimal role assignments," in *Proceedings of INFOCOM '2002*, pp. 1587–1596, IEEE Computer Society, June 2002.

[12] Biagioni, E. and Bridges, K., "The application of remote sensor technology to assist the recovery of rare and endangered species," *Special Issue on Distributed Sensor Networks for the International Journal of High Performance Computing Applications, Vol. 16, N. 3*, 2002.

[13] Biagioni, E. and Bridges, K., "The application of remote sensor technology to assist the recovery of rare and endangered species," *Special issue on Distributed Sensor Networks for the International Journal of High Performance Computing Applications*, vol. 16, no. 3, 2002.

[14] Biagioni, E. and Sasaki, G., "Wireless sensor placement for reliable and efficient data collection," in *In Proceedings of the Hawaiii International Conference on Systems Sciences*, 2003.

[15] Blum, R., Kassam, S., and H.V.Poor, "Distributed detection with multiple sensors: Part ii-advanced topics," *Proceedings of the IEEE*, January 1997.

[16] Branicky, M. S., Liberatore, V., and Phillips, S. M., "Networked control system co-simulation for co-design," in *Proceedings of American Control Conference*, 2003.

[17] Brooks, R. R., Griffin, C., and Friedlander, D., "Self-organized distributed sensor network entity tracking," *International Journal of High Performance Computing Applications, Vol. 16, No. 3*, 2002.

[18] Cerpa, A., Elson, J., Estrin, D., Girod, L., Hamilton, M., and Zhao, J., "Habitat monitoring: Appliation driver for wireless communications technology," in *Proceedings of ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, 2001.

[19] Chamberland, J. and Veeravalli, V. V., "Distributed detection in sensor networks," *IEEE on Signal Processing, Vol.51, No.2*, 2003.

[20] Chen, A., Muntz, R., Yuen, S., Locher, I., Park., S., and Srivastava, M., "A support infrastructure for the smart kindergarten," *IEEE Pervasive Computing*, vol. 1, no. 2, 2002.

[21] Chen, G., Branch, J., Pflug, M. J., Zhu, L., and Szymanski, B., "Sense: A sensor network simulator," in *Advances in Pervasive Computing and Networking* (Szymanski, B. K. and Yener, B., eds.), pp. 249–269, Springer, 2004.

[22] Chen, J., Guan, Y., and Pooch, U., "Customizing gpsr for wireless sensor networks," in *Proceedings of the 1st IEEE International Conference on Mobile Ad-hoc and Sensor Systems(MASS'04)*, 2004.

[23] Chen, Q., Lam, K.-Y., and Fan, P., "Comments on "distributed bayesian algorithms for fault-tolerant event region detection in wireless sensor networks"," *IEEE Transactions on Computers, Vol.54, No.9*, September 2005.

[24] Cheung, S. Y., Coleri, S., Dundar, B., Ganesh, S., Tan, C., and Varaiya, P., "Traffic measurement and vehicle classification with a single magnetic sensor," in *The 84th Annual Meeting, Transportation Research Board*, 2005.

[25] Chong, C.-Y. and Kumar, S. P., "Sensor networks: Evolution, opportunities, and challenges," in *In Proceedings of the IEEE*, August 2003.

[26] Dishman, E., "Inventing wellness systems for aging in place," *IEEE Computer Magazine*, vol. 37, no. 5, 2004.

[27] Doherty, L., Warneke, B. A., Boser, B., and Pister, K. S. J., "Energy and performance considerations for smart dust," *International Journal of Parallel and Distributed Sensor Networks*, December 2001.

[28] Doolin, D. M. and Sitar, N., "Wireless sensors for wildfire monitoring," in *In Proceedings of SPIE Symposium on Smart Structures and Materials/ NDE 2005*, 2005.

[29] Duarte-Melo, E. and Liu, M., "Analysis of energy consumption and lifetime of heterogeneous wireless sensor networks," in *Proceedings of IEEE Globecom'02*, 2002.

[30] Dunkels, A., Alonso, J., and Voight, T., "Making tcp/ip viable for wireless sensor networks," in *European Workshop on Wireless Sensor Networks (EWSN)*, 2004.

[31] Estrin, D., Govindan, R., and Heidemann, J., "Scalable coordination in sensor networks," Tech. Rep. Technical Report 99-692, University of Southern California, 1999.

[32] Estrin, D., Govindan, R., Heidemann, J., and Kumar, S., "Next century challenges: Scalable coordination in sensor networks," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'99)*, 1999.

[33] Gao, T., Greenspan, D., and Welsh, M., "Vital signs monitoring and patient tracking over a wireless network," in *In Proceedings of the 3rd International Conference on Information Communication Technologies in Health (ICICTH'05)*, 2005.

[34] Gao, T., Greenspan, D., Welsh, M., Juang, R. R., and Alm, A., "Vital signs monitoring and patient tracking over a wireless network," in *Proceedings of the 27th International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS'05)*, 2005.

[35] Gifford, D. K., "Weighted voting for replicated data," in *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, 1979.

[36] He, T., Stankovic, J. A., Lu, C., and Abdelzaher, T. F., "Speed: A stateless protocol for real-time communication in sensor networks," in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS'03)*, 2003.

[37] Heinzelman, W., Sinha, A., Wang, A., and Chandrakasan, A., "Energy-scalable algorithms and protocols for wireless microsensor networks," in *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP '00)*, June 2000.

[38] Heinzelman, W. B., Chandrakasan, A. P., and Balakrishnan, H., "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on Wireless Communications, Vol.1, No.4*, October 2002.

[39] Heinzelman, W. B., Kulik, J. W., and Balakrishnan, H., "Adaptive protocols for information dissemination in wireless sensor networks," in *Proccedings of MOBICOM'99*, 1999.

[40] Heinzelman, W. R., Kulik, J., and Balakrishnan, H., "Adaptive protocols for information dissemination in wireless sensor networks," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'99)*, 1999.

[41] Henriksson, D., Cervin, A., and Arzen, K.-E., "TrueTime: Simulation of control loops under shared computer resources," in *Proceedings of the 15th IFAC World Congress on Automatic Control*, 2002.

[42] HUANG, Y., LOEWKE, K., SCHAAF, K., and NEMAT-NASSER, S., "Localized shm with embedded sensor network," in *Proceedings of the 5th International Workshop on Structural Health Monitoring*, 2005.

[43] IEEE, "Ieee standard 802-11 wireless lan medium access control (mac) and physical layer (phy) specification," *Institute of Electrical and Electronic Engineers*, 1999.

[44] Intanagonwiwat, C., Govindan, R., and Estrin, D., "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, pp. 56–67, August 2000.

[45] Johnson, D. B. and Maltz, D. A., "Dynamic source routing in ad hoc wireless networks," *Mobile Computing*, pp. 153–181, 1996.

[46] Karlof, C. and Wagner, D., "Secure routing in wireless sensor networks: Attacks and countermeasures," in *IEEE International Workshop on Sensor Network Protocols and Applications*, 2003.

[47] Kochhal, M., Schwiebert, L., and Gupta, S., "Role-based hierarchical self organization for wireless ad hoc sensor networks," in *Proceedings of the Second ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'03)*, 2003.

[48] Koushanfar, F., Potkonjak, M., and Sanjiovanni-Vincentelli, A., "Fault tolerance techniques for wireless ad hoc sensor networks," in *Proc of IEEE Sensors*, 2002.

[49] Krishnamachari, B. and Iyengar, S., "Distributed bayesian algorithms for fault-tolerant event region detection in wireless sensor networks," *IEEE Transactions on Computers, Vol.53, No.3*, March 2004.

[50] Kuhn, F., Moscibroda, T., and Wattenhofer, R., "Initializing newly deployed ad hoc and sensor networks," in *Proceedings of the 10th annual international conference on Mobile computing and networking (MobiCom'04)*, 2004.

[51] Kulik, J., Heinzelman, W. R., and Balakrishnan, H., "Negotiation-based protocols for disseminating information in wireless sensor networks," *Wireless Networks*, vol. 8, no. 2-3, pp. 169–185, 2002.

[52] Levis, P., Lee, N., Welsh, M., and Culler., D., "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.

[53] Li, D., Wong, K., Hu, Y., and Sayeed, A., "Detection, classification, and tracking of targets," *IEEE Signal Processing Magazine, 19(2):17–30*, 2002.

[54] Liu, J., Perrone, L. F., Nicol, D. M., Liljenstam, M., Elliott, C., and Pearson, D., "Simulation modeling of large-scale ad-hoc sensor networks," in *European Simulation Interoperability Workshop*, 2001.

[55] Liu, Y. and Ni, L. M., "Location-aware id assignment in wireless sensor networks," in *Proceedings of the The 3rd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS'06)*, 2006.

[56] Lj.Bodrozic, D.Stipanicev, and M.Stula, "Agent based data collecting in a forest fire monitoring system," in *In Proceedings of the IEEE Internacional Conference on Software in Telecommunications and Computer Networks (SoftCom'06)*, 2006.

[57] Luo, X., Dong, M., and Huang, Y., "On distributed fault-tolerant detection in wireless sensor networks," *IEEE Transactions on Neural Networks, to appear*, 2005.

[58] Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D., and Anderson, J., "Wireless sensor networks for habitat monitoring," in *Proceedings of*

ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02), 2002.

[59] Makhorin, A., "GLPK: GNU Linear Programming Kit." Software on-line: www.gnu.org/software/glpk/glpk.html, Accessed in January 2006. Moscow Aviation Institute, Russia.

[60] Mallanda, C., Else, S., Suri, A., Kunchkarra, V., Iyengar, S., Kannan, R., and Durresi, A., "Simulating wireless sensor networks with omnet++," *IEEE Computer*, 2005.

[61] Mallanda, C., Kulshrestha, A., Kannan, R., Durresi, A., and Iyengar, S., "Ebrp: Energy band based routing protocol for wireless sensor networks," in *Proceedings of the of International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP'04)*, 2004.

[62] Maroti, M., "Directed flood-routing framework for wireless sensor networks," in *Proceedings of Middleware 2004,*, 2004.

[63] McCanne, S. and Floyd, S., "The LBNL network simulator." Software on-line: http://www.isi.edu/nsnam, Accessed in January 2004. Lawrence Berkeley Laboratory.

[64] Meguerdichian, S., Slijepcevic, S., Karayan, V., and Potkonjak, M., "Localized algorithms in wireless ad-hoc networks: location discovery and sensor exposure," in *Proceedings of MOBIHOC 2001, pg. 106-116.*, 2001.

[65] Milenkovic, A., Otto, C., and Jovanov, E., "Wireless sensor networks for personal health monitoring: Issues and an implementation," *Computer Communications (Special issue: Wireless Sensor Networks: Performance, Reliability, Security, and Beyond)*, vol. 29, no. 13, 14, 2006.

[66] Motegi, S., Yoshihara, K., and Horiuchi, H., "Implementation and evaluation of on-demand address allocation for event-driven sensor network," in *Proceeding of the Symposium on Applications and the Internet (SAINT'05)*, 2005.

[67] on Health Care, T. N. C., "Health insurance cost." http://www.nchc.org/facts/cost.shtml, Accessed in March 2007.

[68] Opnet Technologies, "Opnet." Software on-line: http://www.opnet.com, Accessed in August 2006.

[69] Ould-Ahmed-Vall, E., Blough, D. M., Riley, G. F., and Heck, B. S., "Distributed unique global id assignment for sensor networks," in *Proceedings of 2nd IEEE Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, 2005.

[70] Ould-Ahmed-Vall, E., Heck, B. S., and Riley, G. F., "Simulation of large-scale networked control systems using GTSNetS," in *Springer's Lecture Notes in Control and Information Sciences, Vol. 331* (Antsaklis, P. J. and , P. T., eds.), Springer, 2006.

[71] Ould-Ahmed-Vall, E., Riley, G. F., Heck, B. S., and Reddy, D., "Simulation of large-scale sensor networks using GTSNetS," in *Proceedings of Eleventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'05)*, 2005.

[72] PalChaudhuri, S., Du, S., Saha, A. K., and Johnson, D. B., "Treecast: A stateless addressing and routing architecture for sensor networks," in *Proceedings of the 4th IPDPS International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN 2004)*, 2004.

[73] Park, S., Savvides, A., and Srivastava, M. B., "SensorSim: A simulation framework for sensor networks," in *Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2000)*, pp. 104–111, 2000.

[74] Parker, L., Birch, B., and Reardon, C., "Indoor target intercept using an acoustic sensor network and dual wavefront path planning," in *Proceedings of IEEE International Symposium on Intelligent Robots and Systems (IROS '03)*, 2003.

[75] Pattem, S., Poduri, S., and Krishnamachari, B., "Energy-quality tradeoffs for target tracking in wireless sensor networks," in *2nd Workshop on Information Processing in Sensor Networks*, April 2003.

[76] Perkins, C. E., Royer, E. M., Das, S. R., and Marina, M. K., "Performance comparison of two on-demand routing protocols for ad hoc networks," *IEEE Personal Communications*, vol. 8, pp. 16–28, Feb 2001.

[77] Perkins, C. E. and Royer, E. M., "Ad hoc on-demand distance vector routing.," in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90–100, Feb 1999.

[78] Perrone, L. F. and Nicol, D. M., "A scalable simulator for TinyOS applications," in *Proceedings of the 2002 Winter Simulation Conference*, 2002.

[79] Petriu, E. M., Georganas, N. D., Petriu, D. C., Makrakis, D., and Groza, V. Z., "Sensor-based information appliances," *IEEE Instrumentation and Measurement Magazine*, vol. 3, 2000.

[80] Philipose, M., Consolvo, S., Choudhury, T., Fishkin, K., Perkowitz, M., Smith, I., Fox, D., Kautz, H., and Patterson, D., "Fast, detailed inference of diverse daily human activities," in *Sixth International Conference on Ubiquitous Computing (UbiComp'04)*, 2004.

[81] Pister, K., "My view of sensor networks in 2010." Website: http://robotics.eecs.berkeley.edu/ pister/SmartDust/in2010, Accessed in February 2006.

[82] Polley, J., Blazakis, D., McGee, J., Rusk, D., and Baras, J., "Atemu: A fine-grained sensor network simulator," in *Proceedings of the First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, 2004.

[83] Qi, H., Kurganti, P., and Xu, Y., "The development of localized algorithms in wireless sensor networks," *Sensors, 2, 286-293.*, 2002.

[84] Qi, H., Wang, X., Iyengar, S. S., and Chakrabarty, K., "High performance sensor integration in distributed sensor networks using mobile agents," *International Journal of High Performance Computing Applications, Vol. 16, No. 3, pages: 325–335*, 2002.

[85] Raghunathan, V., Schurgers, C., Park, S., and Srivastava, M. B., "Energy-aware wireless microsensor networks," *IEEE Signal Processing Magazine*, vol. 19, pp. 40–50, March 2002.

[86] Reddy, D., Riley, G. F., Larish, B., and Chen, Y., "Measuring and explaining differences in wireless simulation models," in *Proceedings of 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS06)*, 2006.

[87] Riley, G. F., "The Georgia Tech Network Simulator," in *Proceedings of Workshop on Models, Methods, and Tools for Reproducible Network Research (MoMeTools)*, Aug 2003.

[88] Riley, G. F., "Large-sacle network simulation with GTNetS," in *Proceedings of the 2003 Winter Simulation Conference*, 2003.

[89] Savvides, A., Han, C.-C., and Strivastava, M. B., "Dynamic fine-grained localization in ad-hoc networks of sensors," in *Proceedings of MOBICOM'01*, 2001.

[90] Scalable Network Technologies, "Qualnet." Software on-line: http://www.scalable-networks.com/, Accessed in August 2006.

[91] Schurgers, C., Kulkarni, G., and Srivastava, M. B., "Distributed on-demand address assignment in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems, Vol.13, No.10*, October 2002.

[92] Schwiebert, L., Gupta, S. K. S., and Weinmann, J., "Research challenges in wireless networks of biomedical sensors," in *Mobile Computing and Networking*, pp. 151–165, 2001.

[93] Schwiebert, L., Gupta, S. K. S., and Weinmann, J., "Research challenges in wireless networks of biomedical sensors," in *In Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom'01)*, 2001.

[94] Shih, E., Cho, S., Ickes, N., Min, R., Sinha, A., Wang, A., and Chandrakasan, A., "Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks," in *7th Annual ACM SIGMOBILE Conference on Mobile Computing and Networking*, 2001.

[95] SIG, B., "Bluetooth specification version 1.1." On-line: http://www.bluetooth.com, Accessed in March 2007.

[96] Singh, M. and Prasanna, V. K., "Distributed collaborative computation in wireless sensor systems," in *Handbook of Algorithms for Wireless Networking and Mobile Computing* (Boukerche, A., ed.), Chapman and Hall/CRC, 2006.

[97] Smith, J. R., "Distributing identity," *IEEE Robotics and Automation Magazine, Vol.6, No.1*, March 1999.

[98] Sobeih, A., Chen, W. P., Hou, J. C., Kung, L. C., Li, N., Lim, H., Tyan, H. Y., and Zhang, H., "J-sim: A simulation environment for wireless sensor networks.," in *Annual Simulation Symposium*, 2005.

[99] Sohrabi, K., Ailawadhi, V., Gao, J., and Pottie, G., "Protocols for self organization of a wireless sensor network," *Personal Communication Magazine*, vol. 7, 2000.

[100] Steere, D. C., Baptista, A., McNamee, D., Pu, C., and Walpole, J., "Research challenges in environmental observation and forecasting systems," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, pp. 292–299, 2000.

[101] Sundresh, S., Kim, W., and Agha, G., "SENS: A sensor, environment and network simulator," in *The 37th Annual Simulation Symposium (ANSS37)*, April 2004.

[102] Szewczyk, R., Osterweil, E., Polastre, J., Hamilton, M., Mainwaring, A., and Estrin, D., "Application driven systems research: Habitat monitoring with sensor networks," *Communications of the ACM Special Issue on Sensor Networks*, vol. 46, no. 6, 2004.

[103] Tanenbaum, A. S., *Computer Networks*. Englewood Cliffs, 1989.

[104] Titzer, B. L., Lee, D., and Palsberg, J., "Avrora: Scalable sensor network simulation with precise timing," in *Proceedings of the 4th International Conference on Information Processing in Sensor Networks (IPSN05)*, 2005.

[105] Varshney, P. K., *Distributed Detection and Data Fusion*. Springer, 1996.

[106] Vasu, B., Varshney, M., Rengaswamy, R., Marina, M., Dixit, A., Aghera, P., Srivastava, M., and Bagrodia, R., "Squalnet: a scalable simulation framework for sensor networks," in *Proceedings of the 3rd international conference on Embedded networked sensor systems (SenSys'05)*, 2005.

[107] Viswanathan, R. and Varshney, P. K., "Distributed detection with multiple sensors: Part i-fundamentals," *Proceedings of the IEEE, Vol.85, No.1*, 1997.

[108] Wang, H., Elson, J., Girod, L., Estrin, D., and Yao, K., "Target classification and localization in habitat monitoring," in *Proceedings of ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, 2001.

[109] Wang, X., Xing, G., Zhang, Y., Lu, C., Pless, R., and Gill, C., "Integrated coverage and connectivity configuration in wireless sensor networks," in *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys'03)*, 2003.

[110] Werner-Allen, G., Johnson, J., Ruiz, M., Lees, J., and Welsh, M., "Monitoring volcanic eruptions with a wireless sensor network," in *Proceedings of European Workshop on Sensor Networks (EWSN'05)*, 2005.

[111] Xing, G., Lu, C., Pless, R., and O'Sullivan, J. A., "Co-grid: An efficient coverage maintenance protocol for distributed sensor networks," in *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN'04)*, 2004.

[112] Y. Yu, R. Govindan, D. E., "Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks," Tech. Rep. TR-01-0023, UCLA/CSD, 2001.

[113] Ye, W., Heidemann, J., and Estrin, D., "An energy-efficient mac protocol for wireless sensor networks," in *INFOCOM*, 2002.

[114] Zeng, X., Bagrodia, R., and Gerla, M., "Glomosim: a library for parallel simulation of large-scale wireless networks," in *Proceedings of the 12th Workshop on Parallel and Distributed Simulations (PADS'98)*, 1998.

[115] Zhang, X. and Riley, G. F., "Bluetooth simulations for wireless sensor networks using GTNetS," in *Proceedings of 12th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS04)*, 2004.

[116] Zhao, F., Shin, J., and Reich, J., "Information-driven dynamic sensor collaboration for tracking applications," *IEEE Signal Processing Magazine*, March 2002.

[117] Zhou, H., Mutka, M. W., and Ni, L. M., "Reactive id assignment for wireless sensor networks," *International Journal of Wireless Information Networks, Vol.13, No.4*, October 2006.

VITA

ElMoustapha Ould-Ahmed-Vall was born and grew up in Mauritania. He received his Diplôme d'Ingénieur in computer science from the Université de Technologie de Compiègne, France in 2003. During his undergraduate studies, ElMoustapha spent two semesters as an exchange student at the University of Pennsylvania, USA between 2000 and 2001. He received his Master of Science (M.S.) and Doctor of Philosophy (Ph.D.) degrees in Electrical and Computer from the Georgia Institute of Technology, USA in 2004 and 2007, respectively. He has publications on topics including sensor network simulation and modeling, ID assignment for sensor networks, fault-tolerant detection using sensor networks and computer architecture performance modeling and analysis.

ElMoustapha received the nation-wide best grade in the Mauritanian Baccalauréat exam in 1997. He served on the board of directors of the Université de Technologie de Compiègne, France between 2001 and 2003. He also served on the advisory board of the CS department during the same period.

Between June 2001 and February 2002, ElMoustapha worked as an intern with the Internationalization and Software Testing Team at Manugistics in Rockville, MD, USA. He also interned with SchlumbergerSema between July 2002 and January 2003 in Paris, France. In 2003, he joined the department of Electrical and Computer Engineering at Georgia Tech as a Research Assistant. In 2006, he spent 8 months doing a research internship on computer architecture performance analysis at Intel Corporation in Phoenix, AZ, USA. ElMoustapha will be going back to Intel as a full-time employee in May 2007.